VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2021 Bc. Matej Pancák

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

SYSTÉM SLEDOVÁNÍ ZMĚN V PASIVNÍCH OPTICKÝCH SÍTÍCH
SYSTEM FOR MONITORING CHANGES IN PASSIVE OPTICAL NETWORKS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Matej Pancák**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Martin Holík**
SUPERVISOR

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Telekomunikační a informační technika**

Ústav telekomunikací

Student: Bc. Matej Pancák ***ID:*** 195409
Ročník: 2 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Systém sledování změn v pasivních optických sítích

POKYNY PRO VYPRACOVÁNÍ:

Prozkoumejte možnosti technologie Apache Kafka pro zpracování dat provozu z optických sítí typu GPON. Za účelem dalšího zpracování prozkoumejte strukturu GPON rámců a navrhněte, jaká data by měla být za účelem analýzy provozu dále zpracovávána a prozkoumávána. Příchozí data vhodně roztřiďte a připravte pro další zpracování. Praktickým výstupem práce je vytvořit řešení založené na technologii Apache Kafka a sérii consumerů implementovaných v jazyce Python, kteří budou zajišťovat analýzu dalšího provozu. Součástí práce by měl být i návrh včetně popisu formátu dat ze vstupu do Kafky a výstupu z jednotlivých consumerů. Na závěr práce zhodnoťte použití Vašeho řešení i pro současná doporučení pasivních optických sítí.

DOPORUČENÁ LITERATURA:

[1] D. Hood and E. Trojer, Gigabit-capable passive optical networks. Hoboken: Wiley, 2011

[2] TIŠNOVSKÝ, Pavel, 2019. Použití nástroje Apache Kafka v aplikacích založených na mikroslužbách. Root.cz [online]. [cit. 2020-09-07]. Dostupné z:

https://www.root.cz/clanky/pouziti-nastroje-apache-kafka-v-aplikacich-zalozenych-na-mikrosluzbach/

Termín zadání: 1.2.2021 ***Termín odevzdání:*** 24.5.2021

Vedoucí práce: Ing. Martin Holík

prof. Ing. Jiří Mišurec, CSc.
předseda rady studijního programu

ABSTRAKT

V rámci tejto diplomovej práce bol navrhnutý a implementovaný systém na monitorovanie udalostí v pasívnych optických sieťach, konkrétne v sieťach GPON. Hlavnými technológiami využívanými pri implementácií tohoto systému sú Apache Kafka, Docker a programovací jazyk Python. V rámci vytvorenej aplikácie je implementovaných niekoľko filtrov, ktoré zo zachytených rámcov získavajú podstatné informácie z hľadiska analýzy premávky na danej sieti. Výstupom práce je funkčný systém, ktorý získava zo zachytených rámcov informácie o prevádzke na danej sieti a takto získané informácie ukladá v komponente Apache Kafka, kde sú pripravené na ďalšie spracovanie. V práci sú taktiež uvedené príklady ako takto uložené dáta spracovávať, spolu s informáciami o ich význame a štruktúre.

KĽÚČOVÉ SLOVÁ

GPON, Python, KafkaPython, Apache Kafka, Docker, PLOAM

ABSTRACT

This diploma thesis describes a design and implementation of a system for monitoring events in passive optical networks, specifically in GPON networks. The main technologies used in the implementation of this system are Apache Kafka, Docker and the Python programming language. Within the created application, several filters are implemented. This filters obtain essential information from the captured frames in terms of traffic analysis on the given network. The result of the thesis is a functional system that from the captured GPON frames obtains information about the network traffic and stores them in the Apache Kafka, where the stored data is accessible for further processing. The work also provides examples of how to process the stored data, along with information about their meaning and structure.

KEYWORDS

GPON, Python, KafkaPython, Apache Kafka, PLOAM

PANCÁK, Matej. *Systém na monitorovanie zmien v pasívnych optických sieťach*. Brno, 2031, 80 s. Diplomová práca. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedúci práce: Ing. Martin Holík

VYHLÁSENIE

Vyhlasujem, že svoju diplomovú prácu na tému „Systém na monitorovanie zmien v pasívnych optických sieťach“ som vypracoval samostatne pod vedením vedúceho diplomovej práce, s využitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované v práci a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej diplomovej práce ďalej vyhlasujem, že v súvislosti s vytvorením tejto diplomovej práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a/alebo majetkových a som si plne vedomý následkov porušenia ustanovenia § 11 a nasledujúcich autorského zákona Českej republiky č. 121/2000 Sb., o práve autorskom, o právach súvisiacich s právom autorským a o zmene niektorých zákonov (autorský zákon), v znení neskorších predpisov, vrátane možných trestnoprávnych dôsledkov vyplývajúcich z ustanovenia časti druhej, hlavy VI. diel 4 Trestného zákonníka Českej republiky č. 40/2009 Sb.

Brno ....

podpis autora

POĎAKOVANIE

Rád by som sa poďakoval vedúcemu tejto diplomovej práce Ing. Martinovi Holíkovi, za jeho trpezlivosť a cenné rady, ktoré mi behom vypracovávania tejto práce poskytol. Ďalej by som sa chcel poďakovať rodine a priateľom, ktorý ma počas štúdií podporovali.

Obsah

Zoznam obrázkov

Zoznam výpisov

Úvod

Problematika dohľadu nad internetovou premávkou je v posledných rokoch čoraz aktuálnejšia či už z optimalizačných alebo bezpečnostných hľadísk. Neustále zväčšujúce sa siete, narastajúci počet zariadení pripojených do komunikačných sietí, núti administrátorov vyvíjať pokročilé sledovacie a dohľadové systémy.

Táto práca sa zaoberá problematikou monitorovania udalostí nad pasívnymi optickými sieťami typu GPON. Cieľom tejto práce je preskúmať možnosti nasadenia systému na distribúciu správ Apache Kafka v systéme na monitorovanie pasívnych optických sietí, kde dochádza k prúdovému zachytávaniu veľkého množstva dát. Platforma Apache Kafka sa javí ako ideálne riešenie pre ukladanie veľkého množstva dát, vďaka svojej vysokej priepustnosti a flexibilite. Táto platforma zo svojej definície umožňuje vykonávanie paralelných procesov nad uloženými dátami, vďaka tomuto môžeme predpokladať násobný nárast celkovej priepustnosti systému. Toto tvrdenie potvrdzujú aj viaceré nezávislé zdroje popísané v texte.

Úvod práce sa venuje podrobnému popisu architektúr na distribúciu správ. Teoretická časť sa ďalej venuje podrobnému opisu architektúry Apache Kafka, približuje princípy komunikácie s touto platformou, možnosti konfigurácie a spôsoby využitia tejto platformy v reálnych aplikáciách. Následne sú priblížené výhody využitia tejto platformy pri budovaní dynamických a flexibilných systémov a porovnanie s inými streamovacími platformami. Teoretická časť obsahuje taktiež krátku rozpravu o technológií pasívnych optických sietí a popis štruktúry spracovávaných rámcov fyzickej vrstvy. V rámci tejto časti boli taktiež popísané servisné a riadiace správy sietí GPON, ich štruktúra a význam.

V tretej kapitole je bližšie popísaná praktická časť tejto práce, ktorá obsahuje návrh architektúry a popis implementácie systému s využitím technológie Apache Kafka. Systém pozostáva z dvoch hlavných komponentov. Prvý komponent je tvorený Apache Kafka brokerom a Zookeeper serverom. Druhým komponentom je komponent aplikácie implementovanej v jazyku Python. Tieto komponenty tvoria jednotlivé mikroslužby systému. Aplikácia pozostáva z filtrov, ktoré získavajú žiadané dáta z rámcov a získané informácie ukladá pre ďalšie spracovanie. Popis aplikácie a jej súčastí je v texte sprevádzaný krátkymi ukážkami zdrojových kódov, doplnených o komentáre a popis.

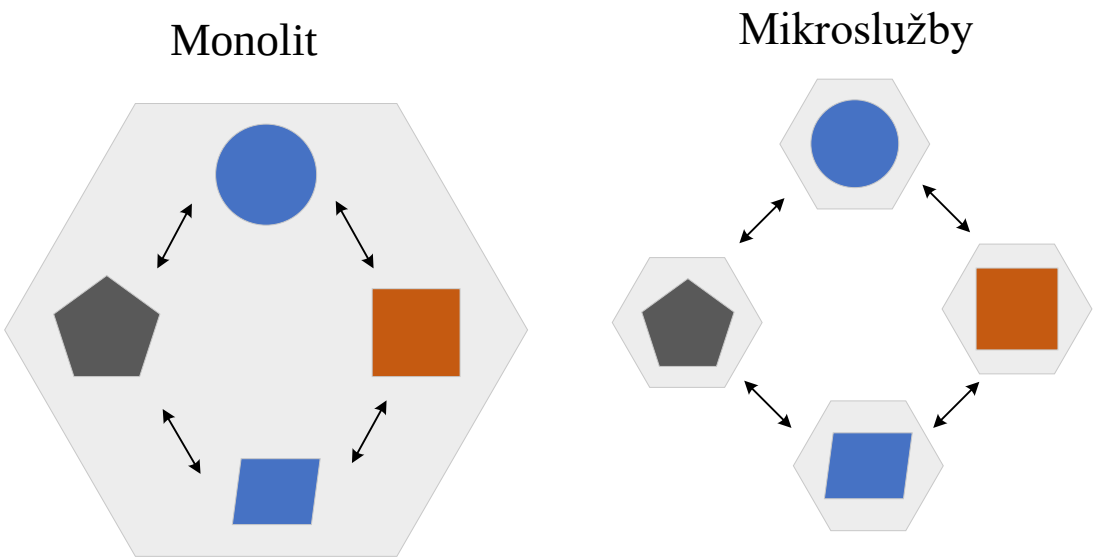
Záver diplomovej práce je krátkym zhrnutím nadobudnutých vedomostí o tejto platforme, získaných počas implementačnej a študijnej časti. V závere je zhodnotený systém a jeho využiteľnosť v novších štandardoch sietí GPON. Výsledkom práce je teda funkčný systém ktorý spracováva rámce pasívnych optických sietí GPON, získané informácie vhodne uchováva za účelom ďalšieho spracovania.

1 Systémy na distribúciu správ

1.1 Architektúra služieb

V dobách minulých bol najčastejšie využívaným prístupom k architektúre aplikácie alebo služby takzvaný *monolitický* princíp viz obr. 1.1. V tomto principiálnom nastavení, boli aplikácie navrhované a následne implementované ako obrovské bloky, združujúce viacero rôznych periférií služby do jedného celistvého bloku resp. monolitu a často komunikovali s jednou rozsiahlou databázou. Tento princíp sa s postupujúcimi technológiami a potrebou expanzie aplikácií a služieb stal zastaralým.

S neustále rastúcou službou, začalo byť takmer nemožné tento monolit meniť a aktualizovať, komplexita monolitu často presahovala možnosti chápania a vnímania programátorov a jeho správa často predstavovala problém. Kvôli týmto skutočnostiam sa myšlienkový prístup k architektúre aplikácií radikálne zmenil. Tento aplikačný monolit sa začal deliť na menšie služby, na takzvané *Mikroslužby* viz obr. 1.1 [1].



Obr. 1.1: Monolit a mikroslužby

Služby a aplikácie postavené na princípe mikroslužieb sú veľmi jednoducho rozšíriteľné o nové funkcionality, správa takýchto systémov je ďaleko jednoduchšia.

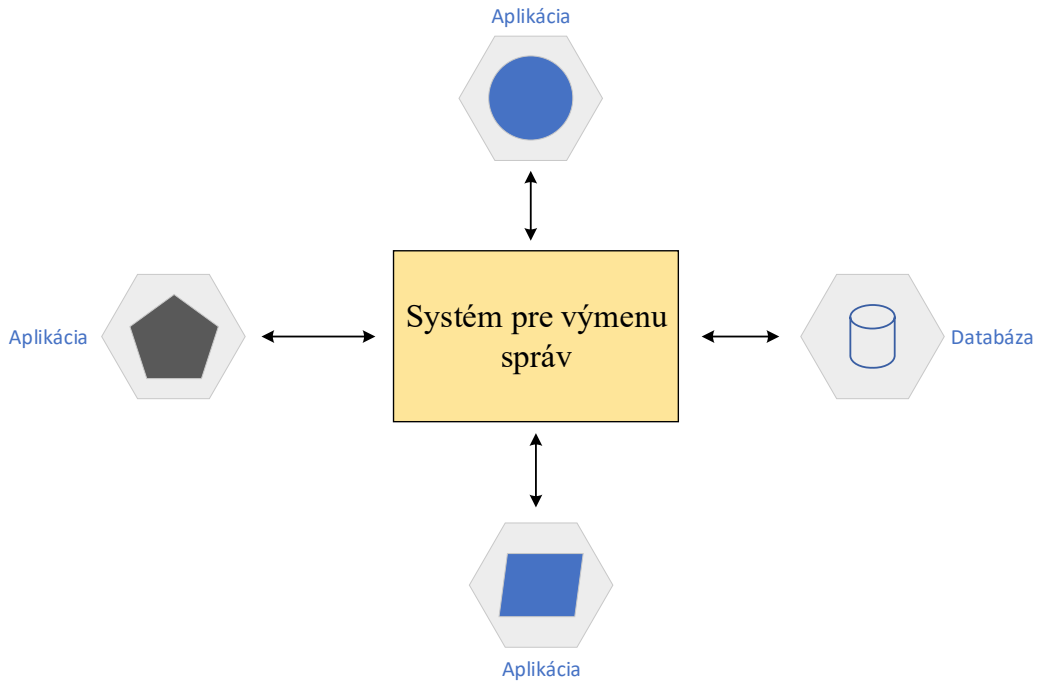
1.1.1 Mikroslužba

Je obsahovo malá a jednoduchá aplikácia, ktorá plní iba jednu funkciu. Mala by byť jednoduchá, ľahko pochopiteľná a malo by byť možné ju publikovať aj samostatne. Aplikácie postavené na architektúre mikroslužieb sú teda zložené z viacerých rozličných mikroslužieb [2]. Jedným z mnohých prístupov ku komunikácii medzi jednotlivými mikroslužbami je napríklad pomocou REST (Representational State Transfer) API (Application Interface), resp. vytváranie takzvaných *point-to-point* spojení. Avšak s postupným rozširovaním systému, pridávaním služieb spracovávajúcich obsah v reálnom čase a pridávaním viacerých mikroslužieb, komunikácia začína byť neprimerane komplexná. Služby začnú byť na sebe neprimerane závislé, čo spôsobí spomalenie a skomplikovanie práce vývojárskym týmom a často aj zníženie priepustnosti systému. Na prekonanie tejto nevýhody, sa nové architektúry zameriavajú na oddelenie odosielateľov správ od príjemcov pomocou asynchrónneho odosielania dát, resp. pomocou asynchrónnej komunikácie a práve túto funkcionalitu prináša Apache Kafka [3]. Ako sa pri čítaní nasledujúceho textu dozvieme, Apache Kafka je omnoho, omnoho viac ako len platformou pre asynchrónne zasielanie správ. Ponúka riešenia vhodné pri spracovaní veľkého množstva dát (Big Data), budovaní streamovacích platforiem a mnoho iných. Apache Kafka je použitá v obrovských a komplexných systémoch spoločností ako LinkedIn, Uber, AirBnb a Netflix, zväčša za účelom monitorovania a poskytovania služieb v reálnom čase [4]. Napokon aj vďaka jej všestrannosti, výkonnosti a flexibilite, bola vybratá ako vhodná platforma pre systém na monitorovanie a spracovávanie dát z pasívnych optických sietí.

1.2 Architektúry systémov na výmenu správ

Predtým, ako prejdeme k jednotlivým technikáliam samotnej Apache Kafka, priblížime si viac systémy pre výmenu správ, ktoré boli zľahka načrtnuté vyššie a sú nedeľnou súčasťou aplikácií postavených na architektúre mikroslužieb. Systém doručovania a výmeny správ a informácií je kľúčovým prvkom každej spoločnosti so širokým portfóliom aplikácií a služieb tvoriacich jednotlivé mikroslužby. V drvivej väčšine prípadov v praxi narazíme nato, že jedna aplikácia alebo služba je závislá práve na výstupoch z inej, partnerskej aplikácie. K dosiahnutiu ucelenej funkcionality takto diverzifikovaného celku, je potrebné aby individuálne aplikácie a služby spolu komunikovali a zdieľali informácie [5]. Táto výmena informácií je sprostredkovaná práve systémami pre výmenu správ (messaging systems). Správami sa rozumejú akékoľvek informácie, logy, dáta a iné výstupy, ktoré jednotlivé aplikácie a služby produkujú. Ako je vyobrazené na obrázku obr. 1.2, aplikácie

a služby postavené na architektúre výmeny správ, zlučujú viacero nezávislých aplikácií, pripojených na spoločný systém na výmenu správ, do ktorého dáta odosielajú a zároveň z neho dáta prijímajú.



Obr. 1.2: Služby s integrovaným systémom pre výmenu správ

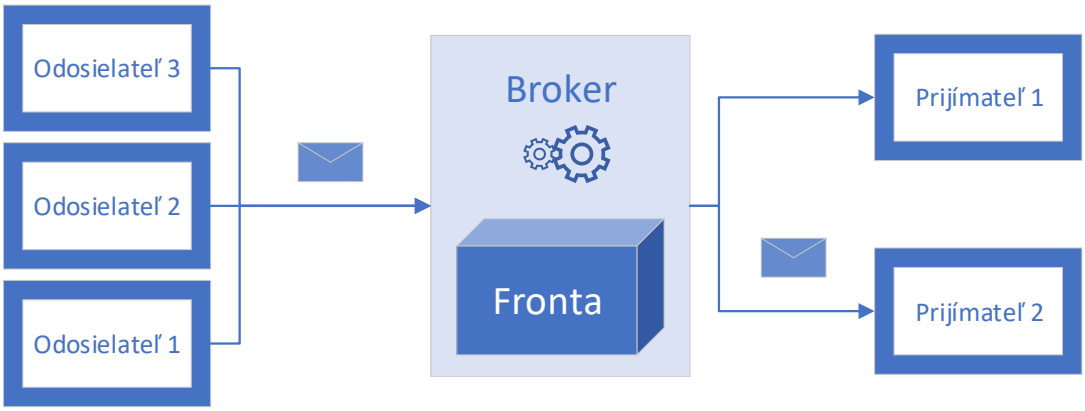
Medzi dve najvýznamnejšie a najviac používané modely výmeny správ môžeme okrem iných zaradiť Publish-Subscribe (PS) a Point-to-Point (PTP) systémy.

1.2.1 Point-to-Point

V PTP modely vymieňania správ sa producenti správ nazývajú *odosielatelia* a konzumenti správ sa nazývajú *prijímatelia*. Výmena správ medzi odosielateľom a prijímateľom prebieha prostredníctvom fronty a to tak, že odosielateľ posiela svoje správy do fronty a prijímateľ z tejto fronty dáta prijíma. Medzi hlavné nevýhody modelu PTP určite patrí fakt, že jednu unikátnu správu môže spracovať len jeden príjemca. Tento fakt je zároveň hlavným rozdiel od iných modelov odosielania správ[5]. Systém PTP vo všeobecnosti používame v prípade, kedy vieme, že správa má byť prijatá práve jedným príjemcom resp. konzumentom. Môžu nastať, prípady, kedy na danej fronte naslúcha viacero konzumentov súčasne, aj v tomto prípade správu obdrží iba jediný z nich. Je nutné poznamenať, že nastať môže aj opačný prípad. Môžeme mať

viacero producentov odosielajúcich dáta do fronty, avšak ich správy môžu byť prijaté len jedným prijímateľom.

Typicky, pri systéme PTP, prijímateľ si namiesto prijatia všetkých správ z odoberanej fronty, vyžiada konkrétnu správu, ktorú odosielateľ poslal do fronty. Fronty podporujúce PTP model výmeny správ, sú typu FIFO (First In First Out) [5]. V takýchto frontách sú správy zoradené v poradí v akom boli do fronty prijaté. Po spracovaní správy, je daná správa odstránená z prvej pozície. Na obrázku obr. 1.3 je vyobrazený príklad modelu PTP, kedy sa na komunikácií podieľajú traja odosielatelia a dvaja prijímatelia.



Obr. 1.3: Point-to-Point model výmeny správ

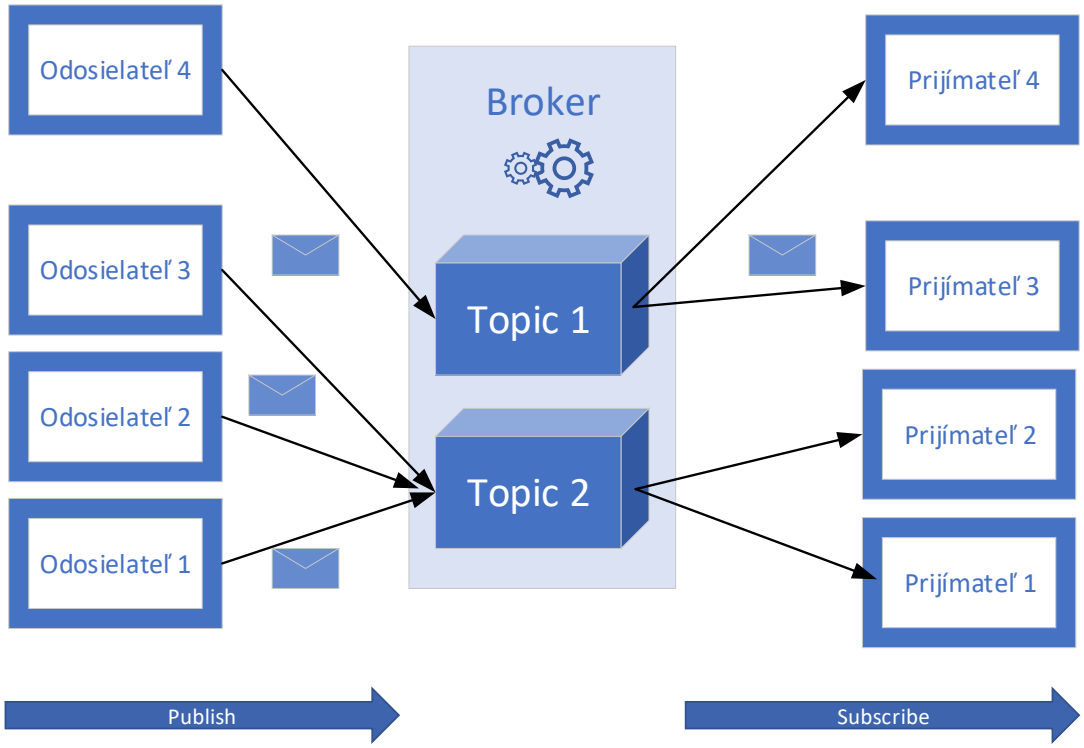
Správy v systémoch PTP sú radené do fronty v poradí takom ako boli prijaté, avšak poradie v akom sú spracovávané záleží na viacerých faktoroch ako: doba životnosti správy, priorita správ a využitie selektovania pri konzumovaní. Odosielatelia a príjemcovia nie sú závislí na časovaní, príjemca môže konzumovať správy nezávisle od producentovho odosielania.

1.2.2 Publish-Subscribe

PS model vymieňania správ je charakteristický tým, že odosielateľ správy neodosiela (publish) priamo k ich príjemcovi. Namiesto toho, odosielateľ správy klasifikuje do rôznych skupín (tém) podľa typu dát alebo podľa iných parametrov, prijímateľ odoberá (subscribe) len skupiny správ, o ktoré má záujem. Prijímateľ má schopnosť prejaviť záujem o danú správu, alebo skupinu správ. Na základe parametrov odberu je v prípade, vygenerovania správy ktorá spĺňa jeho požiadavky, notifikovaný a správa mu je doručená. Každá téma môže mať viacero odberateľov pričom každému z odberateľov je doručená kópia každej správy, táto charakteristická črta pre PS systémy je zároveň hlavným a najdôležitejším rozdielom od systémov PTP [5, 6, 7].

Publish-Subscribe model je teda používaný v prípadoch ak chcem správu doručiť viacerým prijímateľom zároveň. Narozdiel od Point-to-Point modelu kedy aj v prípade, že fronta má viacerých odberateľov (subscribers), správu prijme len jeden [5]. PS systémy pri svojej implementácií často využívajú brokera, centrálny bod do ktorého sú správy publikované a odkiaľ sú správy odoberané.

Na obrázku obr. 1.4 je vyobrazený základný model komunikácie v systémoch Publish-Subscribe. Všetky správy sú zdieľané cez kanál nazývaný téma (topic). Tému si môžeme predstaviť ako frontu správ z ktorej môže odoberať správy viacero konzumentov a viacero producentov do nej môže svoje správy publikovať. Pri tomto modely komunikácie odosielateľ nepozná presnú adresu prijímateľa a ani jeho totožnosť a prijímateľ taktiež nepozná odosielateľa správy, výmena správ tak prebieha v anonymite. Správy doručené do témy sú automaticky pretlačené na všetkých prihlásených odberateľov, bez nutnosti ich vyžiadania.



Obr. 1.4: Publish-Subscribe model výmeny správ

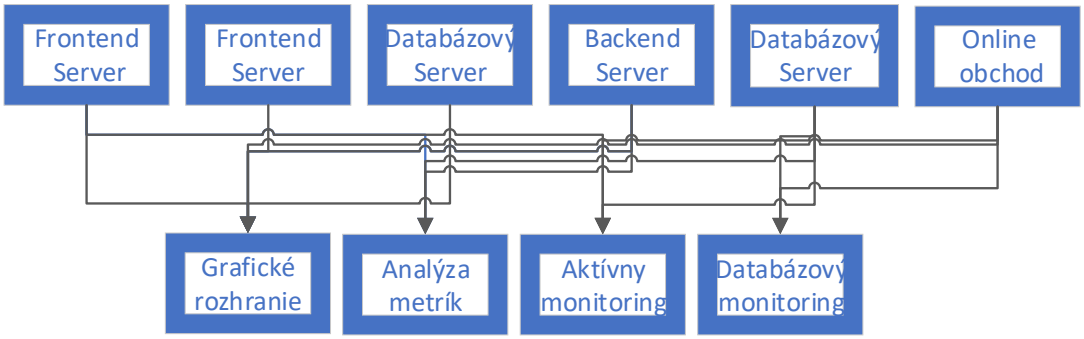
1.2.3 Príklad integrácie systému Publish-Subscribe

V tejto sekcií si priblížime typické príklady využitia systémov postavených na architektúre Publish-Subscribe a prípad kedy je túto architektúru výhodne použiť.

Najbežnejšie prípady využitia sú napríklad:

- **Balansovanie záťaže v sieťovom clustery**[1] – v prípade, že sa nahromadí veľké množstvo správ čakajúcich na spracovanie. Správy sa môžu efektívne rozdeliť medzi viacero spracovávajúcich entít, ktoré môžu jednotlivé správy spracovávať paralelne [8].
- **Distribuovanie notifikácií udalostí** – majme službu, ktorá umožňuje registrácie nových užívateľov a pri každej registrácií nového užívateľa odošle notifikáciu s potrebnými informáciami. Ak chcú služby na opačnej strane prijímať notifikácie, začnú odoberať tému v ktorej sa notifikácie nachádzajú [8].
- **Streamovanie z viacerých služieb a zariadení** – jedna služba môže poskytovať tie isté dáta viacerým službám [8].
- **IoT (Internet of Things)** – viacero senzorov v domácnosti odosiela (publikuje) udalosti a správy do tém tomu odpovedajúcim, z ktorých môže odoberať centrálny systém na spracovávanie dát.

Na nasledujúcom príklade je opísaný spôsob integrácie systému Publish-Subscribe. Majme rastúci počet služieb, ktoré chceme monitorovať a zistené údaje odosielať do externých aplikácií, ktoré následne dané metriky spracovávajú rôznymi spôsobmi. V takomto prípade, ak by sme medzi službami chceli vytvárať priame resp. PTP spojenia, začala by byť správa takéhoto systému náročná a veľmi komplexná viď obr.1.5.

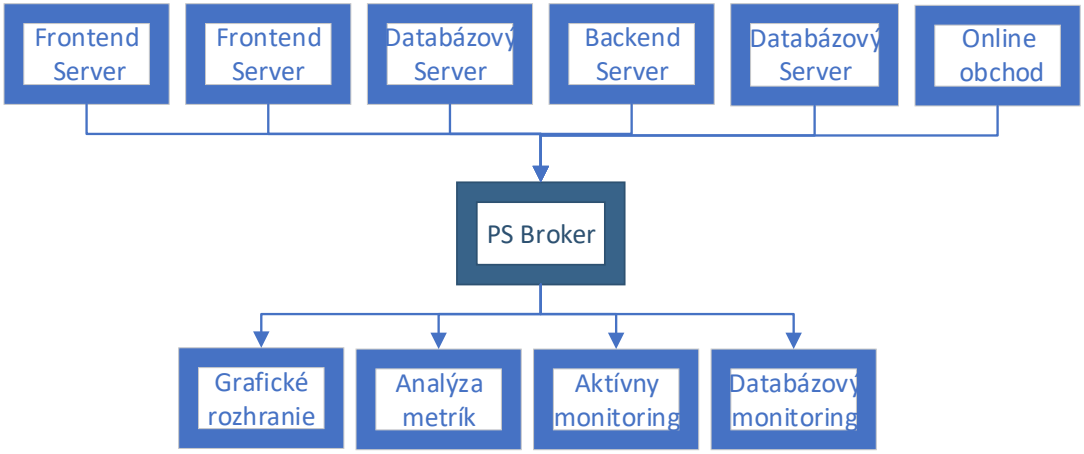


Obr. 1.5: Systém bez integrovaného systému typu Publish-Subscribe

Riešením tohto problému môže byť vytvorenie jedinej aplikácie na báze Publish-Subscribe, ktorá bude prijímať metriky zo všetkých sledovaných služieb a zariadení. Naopak zariadenia, ktoré metriky rôznymi spôsobmi spracovávajú si vyžiadajú dáta

[1] **cluster: je zhluk viacerých sieťových zariadení**

len od tejto aplikácie. Tento spôsob zredukuje komplexnosť architektúry do podoby, ktorú môžeme vidieť na obrázku 1.6.



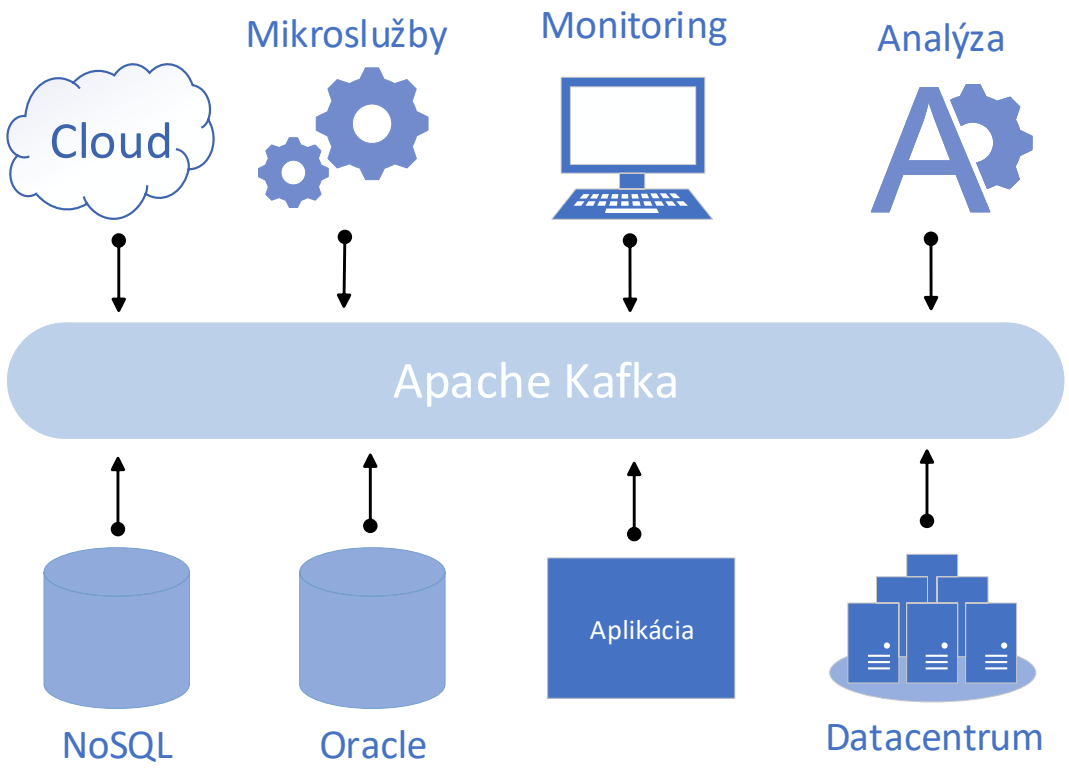
Obr. 1.6: Systém s integrovaným systémom Publish-Subscribe

1.3 Apache Kafka

Apache Kafka je v prvom rade systém pre distribúciu správ typu Publish-Subscribe, avšak najviac používaným prívlastkom pre túto platformu je: „distribuovaná streamovacia platforma“ 1.7. Podobne ako v databázovom systéme sú dáta uložené v odolnej a trvácnej podobe (dáta môžu byť v Apache Kafka uložené po ľubovoľne dlhú dobu), sú ukladané v poradí v akom prišli a môžu byť čítané deterministicky [6]. Všetky tieto funkcie sú dostupné v distribuovanej, vysoko škálovateľnej, odolnej voči chybovosti a bezpečnej forme. Kafka môže byť nasadená napríklad priamo na hardware, vo virtuálnych strojoch, v kontajneroch a samozrejme aj v cloude [9]. Kafka sa primárne používa na spracovávanie dátových tokov a podporu aplikácií fungujúcich v reálnom čase. Kombinuje zasielanie správ, ukladanie a spracovávanie streamov, čo umožňuje ukladanie a analýzu historických údajov a takisto aj údajov v reálnom čase [10].

1.3.1 Princíp fungovania Apache Kafka

Kafka je distribuovaný systém pozostávajúci z komponentov *server* a *klient*, ktoré komunikujú pomocou protokolu TCP (Transmission Network Protocol). Kafka kombinuje kľúčové výhody systémov na výmenu správ, radenie do front zo systému PTP a Publish-Subscribe a poskytuje užívateľom kľúčové benefity oboch systémov [10].



Obr. 1.7: Apache Kafka

Servery

Kafka funguje ako cluster zložený z jedného alebo viacerých serverov, ktoré sa môžu rozprestierať vo viacerých datacentrách, alebo viacerých cloudových regiónoch. Aby bolo možné Kafku implementovať aj v službách s nulovou chybovou toleranciou. Kafku je možné implementovať aj v systémoch s nulovou chybovou toleranciou a to vďaka vysoko flexibilným Kafka clusterom, ktoré môžu byť strategicky rozmiestnené v rozličných oblastiach a v prípade výpadku jedného zo serverov, zastúpia jeho funkciu [9].

Klienti

Klienti umožňujú vytvárať distribuované aplikácie a mikroslužby, ktoré dokážu paralelným, škálovateľným a chybovo tolerantným spôsobom zapisovať a čítať dáta aj v prípadoch zlyhania sieťového pripojenia či hardvérových komponentov.

1.3.2 Hlavné komponenty architektúry Apache Kafka

V tejto sekcií sú opísané vlastnosti kľúčových komponentov Apache Kafka a jej periférií.

Správy

Správy sú vo svojej podstate len polia bajtov, takže dáta obsiahnuté v správach nemajú špecifický formát alebo význam pre Kafku [6]. Správy sú ale kľúčovým prvkom architektúry, všetky udalosti a zmeny naprieč pripojenými zariadeniami generujú správy, ktoré sú následne zapisované resp. prijímané z Kafky. Správa koncepčne pozostáva z kľúča (key), hodnoty (value), časového razítka (timestamp) a možnou súčasťou sú aj hlavičky metadát (metadata headers) [6, 9].

Príklad typickej správy môže vyzerať nasledovne:

- Kľúč správy: „Ján“
- Hodnota správy: „Zaplatil faktúru v hodnote 100 Kč“
- Časové razítko správy: „Aug. 19, 2020 at 3:08 p.m.“

Pre zvýšenie efektivity, sú správy do Kafky zapisované v dávkach (batches). Dávka pozostáva zo správ, ktoré sú určené k zápisu do rovnakej témy (topic) a do rovnakej partície [6].

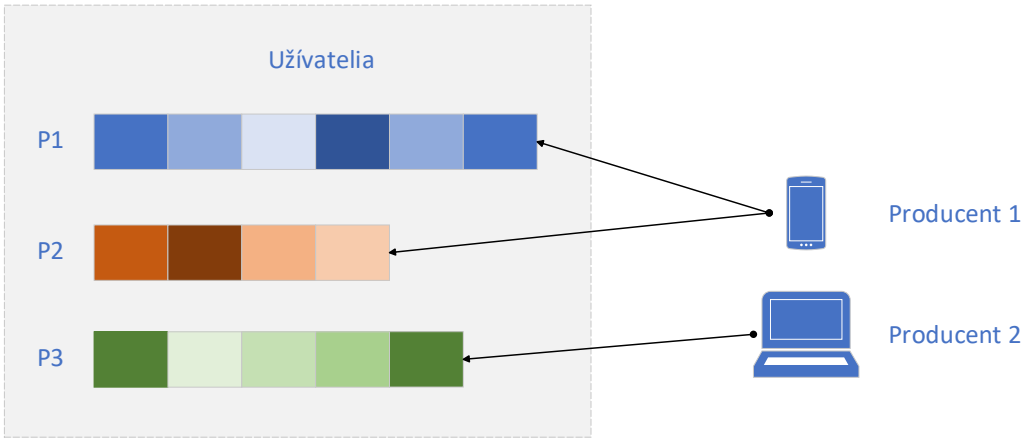
Schémy a štruktúry

Aj keď správy sami o sebe sú z pohľadu Kafky len nezrozumiteľnými poliami bajtov, stále sa odporúča aby na samotný kontext správy bola aplikovaná nejaká štruktúra alebo schéma tak, aby bola ľahko pochopiteľná. Štruktúru obsahu správy môže užívateľ voliť podľa potrieb konzumujúcich aplikácií. Najčastejšie používanými voľbami sú Javascript Object Notation (JSON) a Extensible Markup Language (XML) a to kvôli tomu, že sú jednoducho použiteľné a zrozumiteľné.

Konzistentný formát správ je dôležitý pri využívaní tejto architektúry pretože zapisovanie a čítanie sú dve oddelené činnosti. Ak sa využívajú naprieč Kafkou rozličné formáty správ, aj konzumujúce aplikácie musia byť prispôsobené k tomu, aby rozdielne formáty správ dokázali spracovať [6].

Témy a partície

Správy v Kafke sú kategorizované do tém. Témy sú dodatočne rozdelené do niekoľkých partícií. Túto hierarchiu by sme teda mohli prirovnať ku klasickému súborovému systému, kde téma predstavuje ten najvrchnejší priečinok a jednotlivé partície sú jeho podpriečinky. Správy sú do jednotlivých partícií pridávané na koniec a čítané resp. odoberané od začiatku do konca. Zoradenie správ podľa príchodu nie je garantované naprieč všetkými partíciami, ale len v rámci jednej partície. Partície taktiež ponúkajú možnosti redundancie a škálovateľnosti a to tak, že niektoré partície môžu byť nasadené na iných serveroch v iných častiach siete, čo pohodlne pokryje prípadné hrozby v podobe zlyhania niektorého zo serverov. Na obrázku 1.8 môžeme vidieť príklad zápisu do štyroch partícií témy s názvom „užívatelia“.



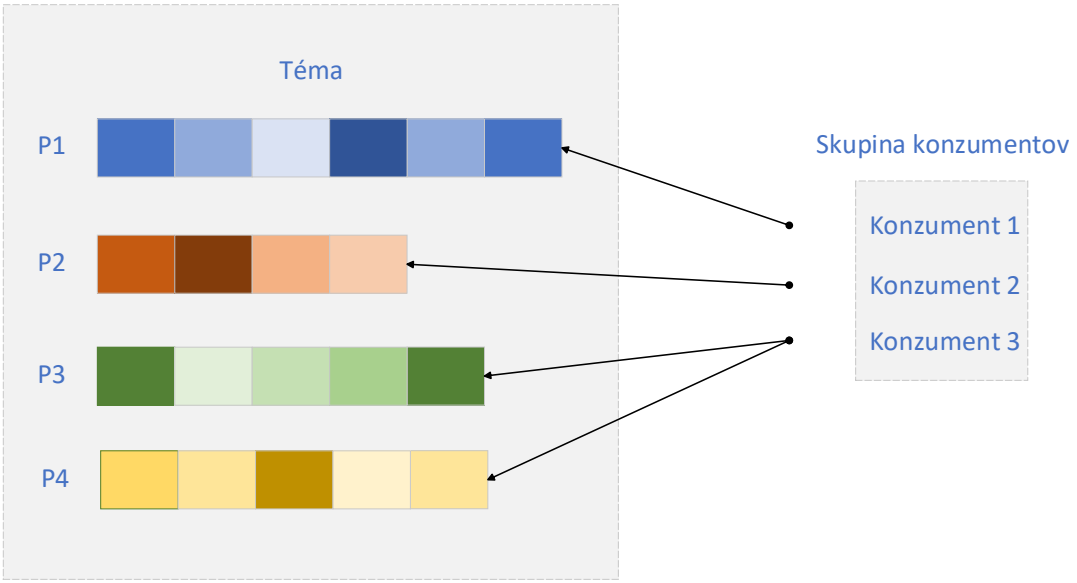
Obr. 1.8: Témy a partície

Producenti a konzumenti

Užívatelia systému Kafka sa nazývajú klienti. Poznáme dva základne typy klientov a to producentov (producents) a konzumentov (consuments). Kafka podporuje taktiež pokročilejšie typy klientov a to napríklad *Kafka Connect API*, ktorá slúži na integráciu už existujúcich databázy. V architektúre Apache Kafka sú výrobcovia a spotrebitelia navzájom úplne oddelení a nezávislí, čo je kľúčovým dizajnovým prvkom na dosiahnutie vysokej škálovateľnosti, ktorou je Kafka známa [6].

Producent je ten typ klientov, ktorý vytvára nové správy a publikuje (zapisuje) ich do Kafky. Vo všeobecnosti sú správy publikované do tém. Producenti sa v predvolenom nastavení vôbec nezaujímajú do akej partície je konkrétna správa zapísaná, zapisovanie správ do partícií rozložia ekvivalentne medzi všetky partície. V niktorých

prípadoch môže producent zapisovať správy priamo do rôznych partícií. Typicky sa odohráva tak že, je vytvorený hash kľúča správy a následne namapovaný na špecifickú partíciu.
Konzument je typ klienta, ktorý správy z Kafky konzumuje resp. číta. V iných Publis – Subscribe systémoch ich tiež nazývame ako subscribers. Konzument môže odoberať jednu alebo viacero tém a správy z nich číta v poradí v akom boli zapísané. Konzument si udržiava záznamy o už spracovaných správach pomocou sledovania offsetu jednotlivých správ. Offset je hodnota typu integer, ktorej hodnota sa sekvenčne zvyšuje po vyprodukovaní novej správy. Každá správa v partícií má unikátny offset. Zapamätaním si offsetu poslednej konzumovanej správy, môže konzumer po znovu obnovení konzumácie pokračovať v konzumácií správ tam kde prestal. Konzument môže pracovať aj ako súčasť *skupiny konzumentov*, pod ktorou sa označuje viacero konzumentov spracovávajúcich jednu tému. V rámci skupiny môže jeden konzument spracovávať len jednu partíciu danej témy. Na obrázku 1.9 môžeme vidieť troch konzumentov v rovnakej skupine, konzumujúcich jednu tému. Mapovanie konzumentov na jednotlivé partície sa často nazýva vlastníctvo (ownership). Týmto spôsobom, môžeme konzumentov horizontálne škálovať tak aby konzumovali témy s obrovským množstvom správ. V prípade výpadku jedného konzumenta, ho ostatní konzumenti v skupine skupiny konzumentov označenej *group – id* zastúpia.

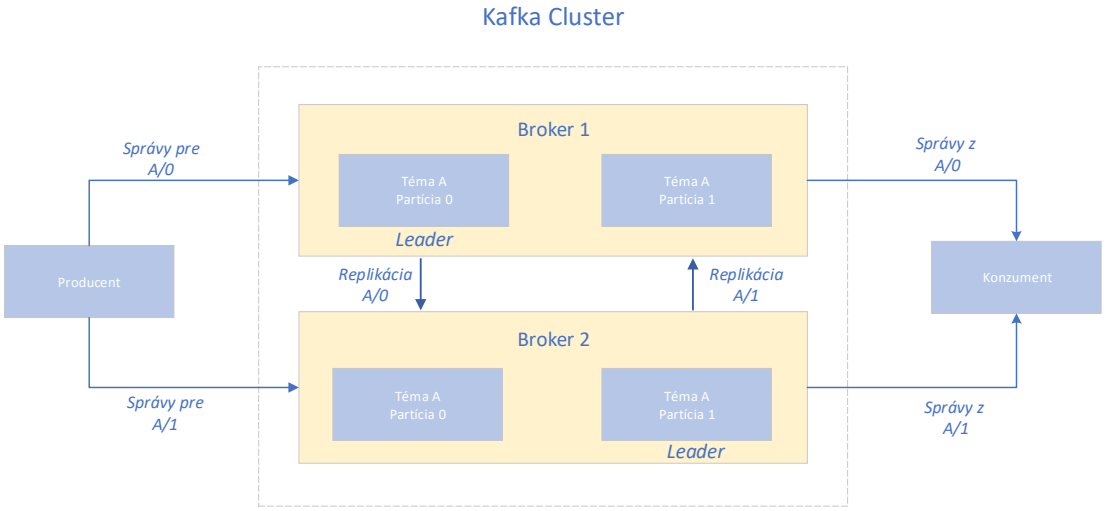


Obr. 1.9: Skupina konzumentov

Brokery a Clustery

Samotný Kafka server sa nazýva broker. Broker prijíma správy z producentov, priraďuje im offset a ukladá ich do pamäte na disk. Taktiež poskytuje služby konzumentom, odpovedá na *fetch* žiadosti na jednotlivé partície a odpovedá im správami, ktoré boli uložené na disk. V závislosti na hardvéry a jeho výkone, môže samostatný broker s ľahkosťou obslúžiť tisíce partícií a milióny správ za sekundu [6].

Kafka brokery sú navrhnuté tak aby vedeli pracovať v rámci clusteru. V rámci clusteru existuje vždy jeden broker s prívlastkom kontrolér. Kontrolér je volený automaticky spomedzi brokerov, ktoré sú aktívne. Kontrolér je zodpovedný za administráciu v rámci clusteru, monitoruje výpadky a priraďuje partície jednotlivým brokerom. Každá partícia je vlastnená jediným brokerom, ktorý je nazývaný lídrom partície. Partícia môže byť priradená viacerým brokerom, čo vyústi do replikácie danej partície, tak ako môžeme vidieť na obrázku 1.10. Toto zabezpečuje potrebnú redundanciu správ v partíciach v prípade výpadku jedného z brokerov.



Obr. 1.10: Kafka Cluster

Kľúčovou vlastnosťou Apache Kafka je retencia, čo schopnosť spoľahlivého a odolného uloženia správ na určité časové obdobie. Nastavenia retenčnej politiky nám umožňujú voliť si dobu uchovávania správ. Príkladom konfigurácie môže byť nastavenie uchovávania správ po určitú dobu napríklad siedmych dní, alebo uchovávať správy kým nebude prekročená istá kapacita na disku. Hneď po prekročení týchto pravidiel sú správy expirované a vymazané. Každá téma je konfigurovaná s jej vlastnou retenčnou politikou [6].

1.4 Porovnanie systémov na distribúciu správ

V tejto sekcií je vypracovaný súhrn aspektov v ktorých Apache Kafka vyniká a na základe ktorých bola práve táto technológia zvolená ako najvhodnejší kandidát na implementáciu v monitorovacom systéme pasívnych optických sietí. V závere tejto sekcie porovnáme Apache Kafka s technológiami Apache Pulsar a RabbitMQ z výkonnostných pohľadov.

1.4.1 Zhrnutie vlastností Apache Kafka

Viacnásobný producenti

Kafka dokáže bez problémov zvládnuť viacerých producentov, či už títo klienti publikujú do veľa tém alebo do rovnakej témy. Vďaka tomu je systém ideálny na agregáciu údajov z mnohých frontendových systémov a na ich konzistentnosť. V systéme na spracovanie informácií z pasívnych optických sietí by mohli byť títo producenti použitý paralelne čím by sa značne zvýšila priepustnosť systému a množstvo zachytených rámcov odoslaných do Kafky.

Viacnásobný konzumenti

Okrem viacerých producentov Kafka podporuje aj viacerých konzumentov, ktorí môžu prijímať ľubovoľný tok správ bez toho, aby si navzájom prekážali. To je v kontraste s mnohými systémami radenia do front, kde keď správu jeden klient spotrebuje, nie je k dispozícii žiadnemu ďalšiemu.

Retencia využívajúca diskové úložisko

Kafka nielen, že zvládne viacero klientov, ale ukladanie správ na disk znamená, že klienti nutne nemusia fungovať v reálnom čase. Napríklad v prípade, že konzument nestíha spracovávať dáta, či už kvôli slabej výkonnosti alebo enormnému nárastu premávky, nemusíme sa obávať straty žiadnych dát. Správy sú uložené na disk spolu s konfiguračnými retenčnými pravidlami a informáciami o uložení.

Škálovateľnosť

Flexibilita Kafky nám umožňuje veľmi jednoducho daný systém rozširovať a tým ho prispôsobiť na spracovávanie akéhokoľvek množstva dát.

Vysoký výkon

Všetky tieto vyššie zmienené vlastni Kafky spoločne vytvárajú streamovaciu platformu, ktorá vyniká excelentnými výkonovými vlastnosťami pri vysokom zaťažení.

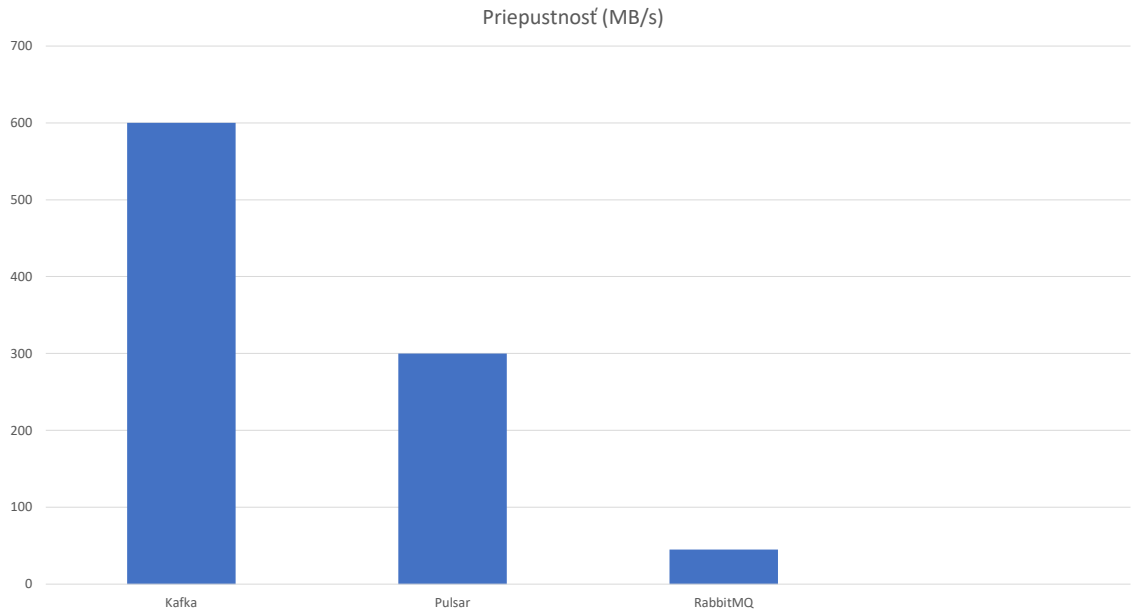
Producenti, konzumenti a brokery môžu byť ľubovoľne rozširované tak aby bez problémov zvládli obrovské streamy správ bez väčších problémov [6].

1.4.2 Porovnanie s technológiami Pulsar a RabbitMQ

Apache Pulsar je open-source systém na výmenu správ. Pôvodne využívaný ako systém na báze front, v posledných vylepšeniach bol rozšírený o možnosti streamovania. RabbitMQ je takisto open-source platforma streamovacia platforma.

Priepustnosť

Ako môžeme vidieť na obrázku 1.11 najvyššiu priepustnosť spomedzi týchto platforiem dosahuje práve Apache Kafka, zapisovaní dosahuje rýchlosti pätnásťnásobne vyššie ako RabbitMQ a dvojnásobne vyššie ako Apache Pulsar [11].



Obr. 1.11: Porovnanie priepustností jednotlivých platforiem [11]

Latencia

Kafka sa vyznačuje najnižšou latenciou pri najvyššej priepustnosti spomedzi porovnávaných platforiem [11].

Vyhodnotenie

Na základe vyššie uvedeného je pre potreby systému na monitorovania pasívnych optických sietí jednoznačne najvhodnejšou technológiou Apache Kafka. Hlavnými

dôvodmi sú vysoká priepustnosť, možnosť ukladania správ na diskové úložisko v prípade výpadku konzumentov a jednoduchá škálovateľnosť.

2 Pasívne optické siete

Pasívna optická sieť PON (Passive Optical Network), predstavuje infraštruktúru optickej prístupovej siete, ktorá využíva k distribúcií optického signálu pasívne optické sieťové komponenty. Pasívna optická sieť nasadzovaná v prístupových sieťach teda medzi koncovým účastníkom a transportnou sieťou. Optická distribučná infraštruktúra je teda tvorená len pasívnymi optickými prvkami ako sú optické vlákna, konektory a pasívne rozbočovače. Nevýhodou pasívnych optických sietí je fakt, že dochádza len k rozbočeniu signálu bez akýchkoľvek úprav ako sú napríklad regenerácia alebo zosilnenie signálu, ktoré sú typické pre aktívne optické prvky. Z hľadiska umiestnenia konečných jednotiek ONU (Optical Netwrok Unit), ONT (Optical Network Terminal) v optických prístupových sieťach a spôsobe ich implementácie rozlišujeme tieto typy optických prístupových sietí [12, 13]:

- FTTC (Fibre To The Curb) – Koncové body siete sú pripojené metalickými káblami.
- FTTB (Fibre To The Building) – Optické vlákno je privedené do budovy, kde sú jednotlivý užívatelia pripojený pomocou vnútornej siete.
- FTTO (Fibre To The Office) – Optické vlákno je privedené až do priestorov užívateľa, ktorý má vysoké nároky na prenosovú kapacitu.
- FTTH (Fibre To The Home) – Optické vlákno je privedené priamo do užívateľských zásuviek.
- FTTN (Fibre To The Node) – Koncovým bodom siete je uzol.
- FTTA (Fibre To The Antenna) – Koncovým bodom siete je anténa.
- FTTO (Fibre To The Desk) – Koncovým bodom siete je priamo stôl užívateľa, optické vlákna sú pripojené priamo do CPE s optickým vstupom.

2.1 Topológia siete

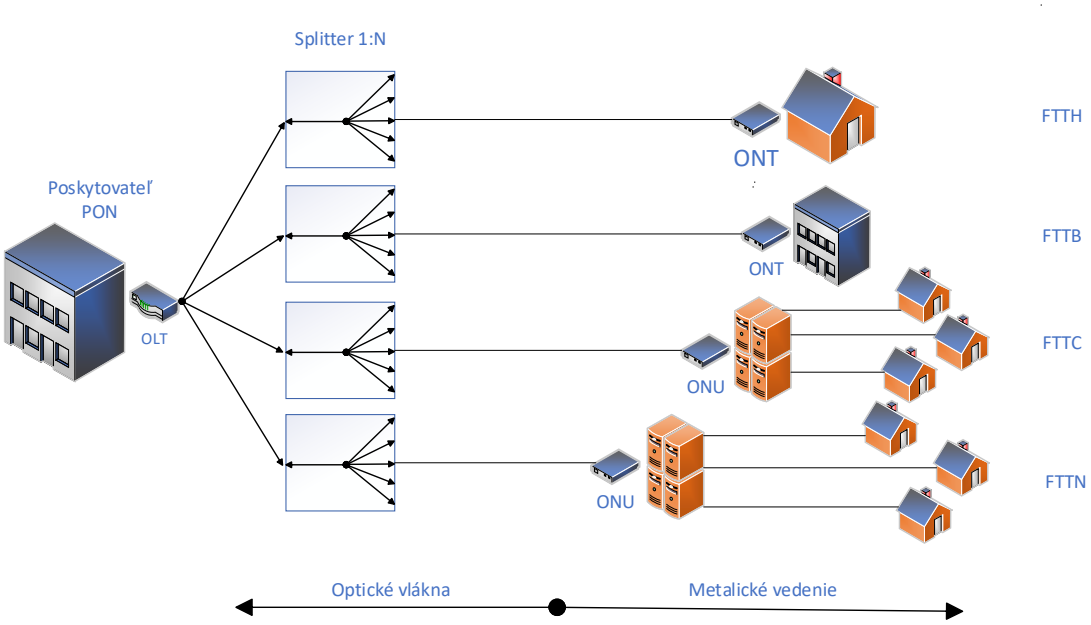
Pasívna optická sieť je realizovaná logickou topológiou typu point-to-multipoint (P2MP), kedy je optický prenosový kanál zdieľaný medzi niekoľkých užívateľov. Topológia PON pozostáva z niekoľkých kľúčových komponentov:

- OLT (Optical Line Terminator) – Je zariadenie, ktoré slúži k zakončeniu linky na strane internetového poskytovateľa. Jeho účelom je konverzia elektrického signálu na optický (a naopak) a multiplexovanie a demultiplexovanie signálu prenášaného pomocou optického vlákna.
- ONU (Optical Network Unit) – Zariadenie, ktoré zakončuje pasívnu optickú sieť na strane zákazníka a stará sa o prevod signálu medzi domácou sieťou

koncových zákazníkov a prístupovou sieťou.

- ONT (Optical Network Terminal) – Je špeciálnym typom ONU. Sprostredkováva služby špecificky pre jedného zákazníka.
- Splitter – Je jednoduché pasívne zariadenie umožňujúce viacerým zákazníkom zdieľať prenosovú šírku jedného optického vlákna.

Každý prvok v sieti do cesty vkladá útlm, tím dochádza aj k obmedzeniu maximálnej vzdialenosti, ktorú je možné optickou sieťou prekonať [12, 13]. Na obrázku 2.1 môžeme vidieť typickú topológiu pasívnych optických sietí.



Obr. 2.1: Topológia PON

2.2 GPON

V roku 2003 bola organizáciou ITU-T (ITU Telecommunication Standardization Sector) schválená špecifikácia G.984.1 GPON (Gigabit Capable PON), ktorá vychádza zo špecifikácií G.983.X. Pre prenos informácií využíva nie len bunky ATM, ale tiež metódu GEM (GPON Encapsulation Method). Tá umožňuje prenášanie paketovo orientovaných služieb ako Ethernet či IP (Internet Protokol). Prenášané rámce majú pevnú dĺžku 125 µs. Obojsmernej premávky môže byť v sieťach PON dosiahnuté buď pomocou dvoch samostatných optických vlákien (jedno pre upstream druhé pre downstream), alebo s využitím vlnového multiplexu (WDD – Wavelength

Division Multiplex). Prenosové rýchlosti sú ponúkané v dvoch variantách a to 1,244 a 2,488 Gbit/s. Novinkou pri štandarde GPON je podpora IPTV [12].

2.2.1 GPON Transmission Convergence (GTC) Layer

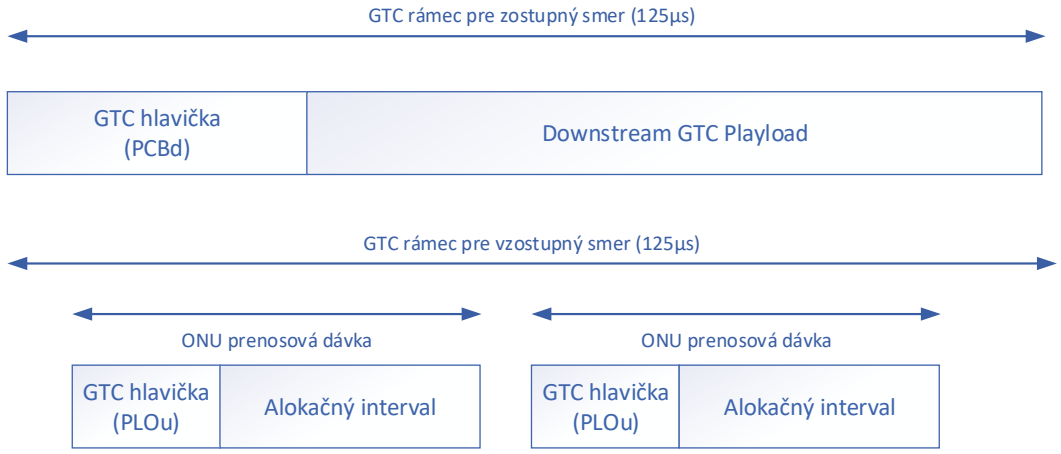
ITU-T odporúčanie G.984.3 popisuje GPON TC vrstvu ako ekvivalent k linkovej vrstve v modele OSI. Táto vrstva je kľúčovou súčasťou technológií na báze pasívnych optických sietí, hlavne architektúr GPON a XG-PON. Táto vrstva ponúka riešenia na unikátne problémy, ktoré predstavujú pasívne optické siete, rovnako ako bežné problémy akéhokoľvek sieťového protokolu. Tieto jedinečné aspekty, ktoré pokrýva GTC vrstva, do značnej miery súvisia so stromovou štruktúrou, teda point-to-multipoint povahou PON [14]:

- Objavovanie doposiaľ nenájdených ONU spôsobom, ktorý nenarušuje existujúcu prevádzku na PON
- Znovupripojenie známych staníc do siete PON po rôznych zlyhaniach a systémových chybách
- Vysporiadanie sa s faktom, že rôzne ONU sú v rozličných vzdialenostiach od OLT, teda majú rôzne propagačné oneskorenie
- Definovanie štruktúry mapovania užitočnej záťaže (payload) s cieľom zabezpečiť celkovú efektivitu prenosu
- Ochrana PON siete od ONU, ktoré svojím chovaním môžu ohrozovať prenos a bezpečnosť siete

Ďalšími oblasťami, ktoré ale nie sú typické pre pasívne optické siete, ale taktiež sa nimi zaoberá GTC vrstva sú [14, 16]:

- Zabezpečenie spojenia medzi OLT a ONU pred odpočúvaním, nedovolenou manipuláciou a krádežou služby.
- Meranie kvality spojenia z pohľadu optických parametrov, ale aj z pohľadu chybovosti
- Zlepšenie vnútornej kvality optického spojenia pomocou FEC
- Podpora signalizácie a kontroly kanálov PON spojenia v rozsahu od jednoduchej bitovo orientovanej signalizácie až po pokročilé doručovanie štruktúrovaných správ v prípade GPON nazývané PLOAM (Physical Layer Operations And Maitenance)
- Zabezpečenie aktívneho znižovania spotreby energie počas časových slotov s nízkou premávkou

Obrázok 2.2 znázorňuje štruktúru GTC rámca pre zostupný a vzostupný smer komunikácie. Rámec pre zostupný smer pozostáva z dvoch blokov a to PCBd (Physical Control Block downstream) a GTC payload. Rámec pre vzostupný smer je zložený z niekoľkých prenosových dávok. Každá dávka pozostáva z PLOu (Physical Layer Overhead upstream) bloku a jedným alebo viacerými pásmovými alokáciami úzko spätými so špecifickým ALLOC_ID [1] [12, 14].

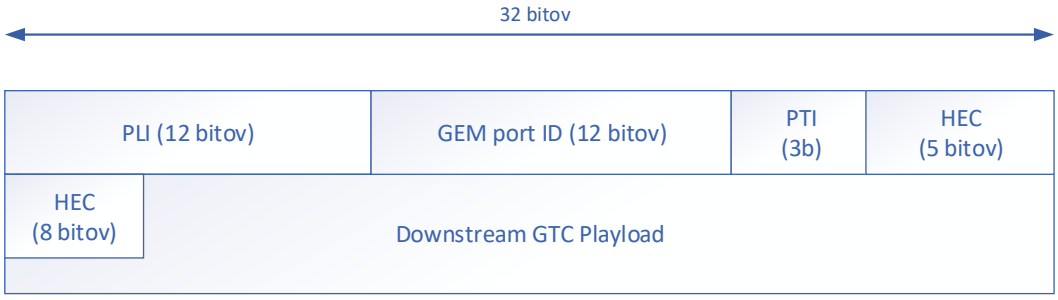


Obr. 2.2: Rámce vrstvy GTC

2.2.2 GEM rámec

Na to aby mohli byť správy prenášané cez sieť GPON, musia byť zapuzdrené do rámcov typu GEM. Každý z GEM rámcov je označený s GEM port ID, čo predstavuje unikátny identifikátor patričného toku v sieti PON. GEM rámce a GEM porty sú viditeľné len medzi OLT a ONU, teda nemajú žiaden hlbší význam z hľadiska celkovej premávky [14]. Štruktúra rámca a jeho najdôležitejšie súčasti sú zobrazené na obrázku 2.3.

[1] ALLOC_ID je jedinečný identifikátor časového slotu na vysielanie vo vzostupnom smere



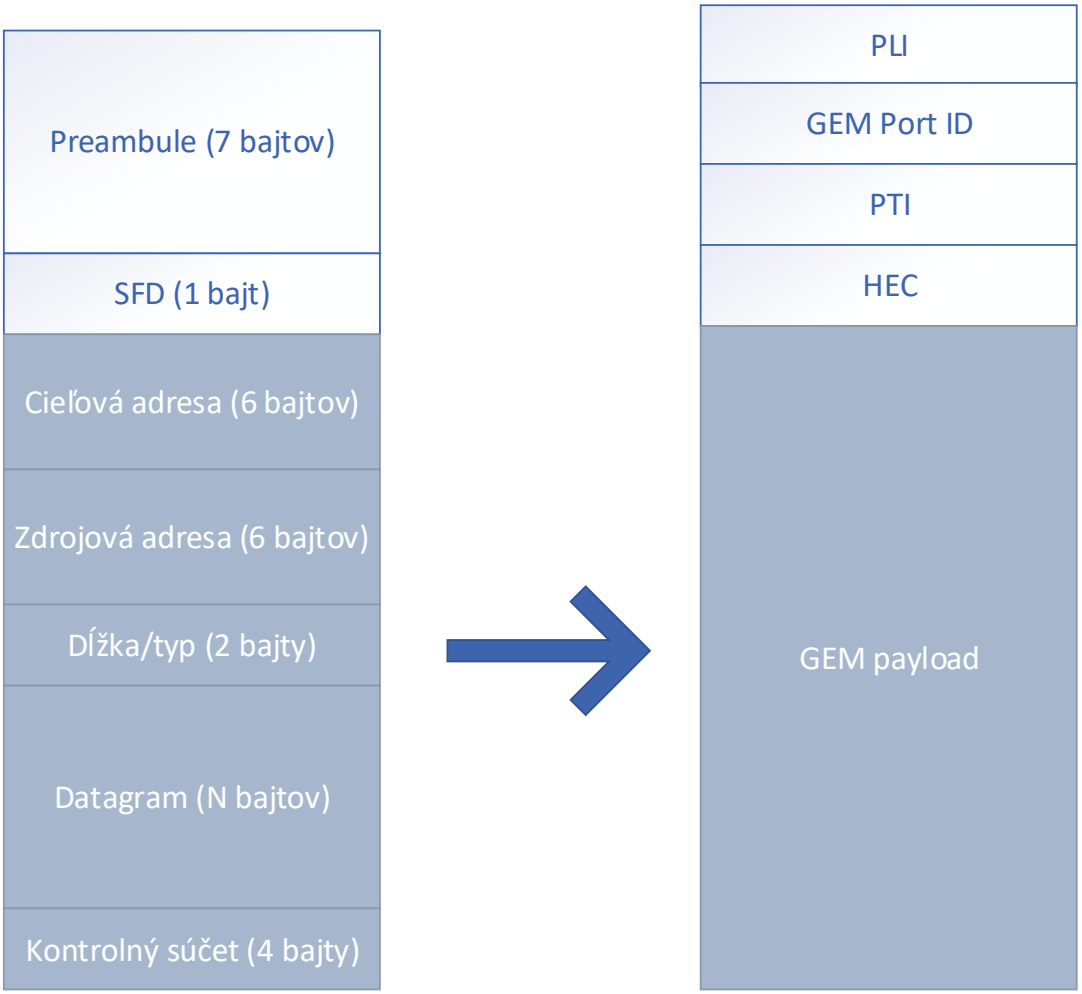
Obr. 2.3: GEM rámec

Význam jednotlivých polí:

- GEM Port ID – GEM port jednoznačne identifikuje dátový tok a uľahčuje mulitplexovanie celkovej premávky. Samotný port je pri štandarde GPON určený z rozsahu 0–4095 [14].
- PLI (Payload Length Indicator) – ukazovateľ dĺžky dát (payload) je pre rámce s premenlivou dĺžkou nevyhnutný. Rozsah je udávaný v bajtoch s rozsahom až do 4095B pri štandarde GPON. GEM rámec je ukončené presne po PLI bajtoch dát[14].
- HEC (Hybrid Error Correction) – GEM rámce sú reťazené jeden za druhým do GTC rámca (GTC payload zo sekcie 2.2.1). Dekodér je schopný úspešne dekódovať nasledujúci GEM rámec len v prípade, ak dokáže správne dekódovať hlavičku aktuálne spracovávaného rámca a najme jeho PLI pole. Preto je v rámci GEM hlavičky nutné využívať kód na opravu a kontrolu chýb nazývaný HEC.
- PTI – identifikuje fragmentáciu SDU (Service Data Unit). PTI tiež odlišuje užívateľské prenosy od prenosov z OAM.

Mapovanie do GEM rámcov

GEM rámce sú prenášané cez GTC protokol transparentným spôsobom. V zostupnom smere sú GEM rámce odosielané z OLT do ONU v poly payload GTC rámca. ONU filtruje prichádzajúce rámce na základe Port ID. Na obrázku 2.4 je znázornený spôsob akým sú do GEM rámcov mapované Ethernetové rámce [14].

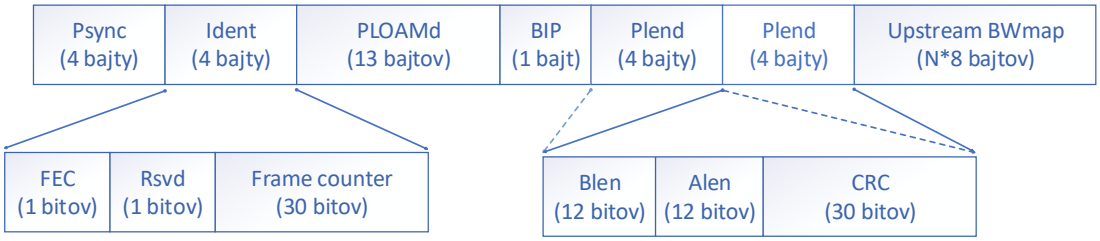


Obr. 2.4: Mapovanie Ethernetového rámca

Ako môžeme vidieť na obrázku, pole Preamble a SFD sú zahodené a zbytok Ethernetového rámca je namapovaný priamo do poľa GEM payload. Jeden Ethernetový rámec môže byť mapovaný do jedného alebo viacerých GEM rámcov [14].

2.2.3 Formát GPON rámcov v zostupnom smere

Ako už bolo spomenuté v kapitole 2.2, štandard GPON pre prenos využíva časový multiplex (TCM — Time Division Multiplex) a rámce v zostupnom smere trvajú presne 125 µs a a majú dĺžku 38 880 bajtov [14]. GTC rámec pre zostupný smer obsahuje hlavičku nazývanú aj PCBd, ktorá pozostáva zo siedmich polí viz obrázok 2.5. Podrobnejší popis jednotlivých polí hlavičky je nižšie.



Obr. 2.5: PCBd

PSYNC

Účelom 32 bitového poľa PSync (Physical Synchronization) je poskytnúť dobre definovanú bitovú sekvenciu, proti ktorej si môže ONU prijímač vytýčiť množstvo bajtov. Tým je zabezpečená synchronizácia fyzickej vrstvy.

IDENT

Toto 32 bitové pole určuje vlastnosti hlavičky:

- FEC – ak je tento bit nastavený na hodnotu 1, rámec je chránený proti chybovému kódovaniu. Na ochranu pre chybami bitov vyžaduje ONU predtým ako rozpozná zmenu, štyri po sebe nasledujúce rámce s rovnakým bitom FEC. To v praxi nepredstavuje žiadny problém s prerušovaním služby, pretože FEC sa nemá meniť dynamicky.
- Rsvd (Reserved) – rezervovaný bit nie je využívaný
- Frame Counter – je to pole dĺžky 30 bitov, ktoré sa v cykloch inkrementuje s každým zostupným rámcom. Jeho primárnym využitím je výmena šifrovacích kľúčov a časová synchronizácia.

PLOAMd (Physical Layer Operation Maitenance)

Pri štandarde GPON sú PLOAM správy dlhé 13 bajtov, každý rámec v zostupnom smere obsahuje takúto správu, v prípade že OLT nemá nič na odoslanie, odošle *no_message* správu.

BIP-8 (Bit Interleaved Parity)

Toto pole obsahuje 8 bajtov prekladaného kódovania. Pomocou týchto bitov vieme overiť všetky bitové chyby vyskytujúce sa v hlavičke GPON rámca od predchádzajúceho poľa BIP s výnimkou paritných bitov. Počítanie BIP je možné použiť aj na monitoring výkonnosti [14].

PLEND (Payload Length Field)

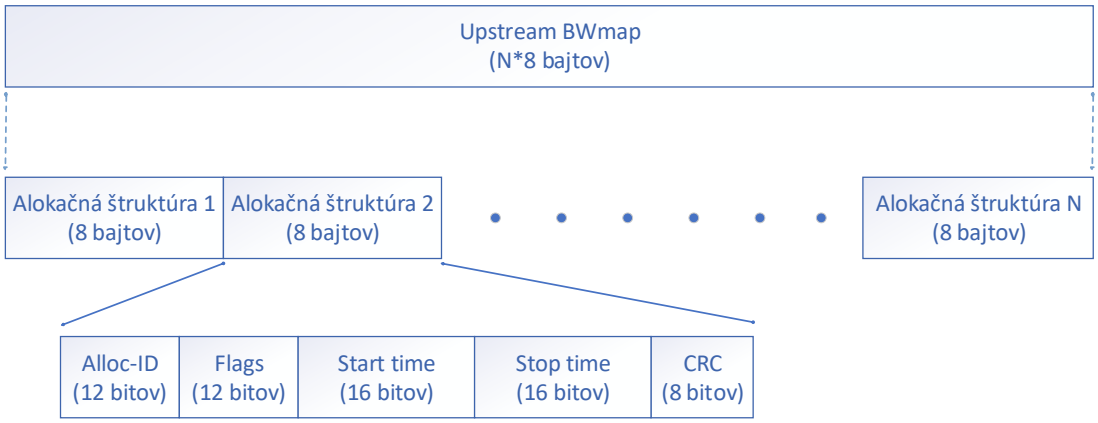
Paradoxne toto pole nedefinuje dĺžku poľa payload. Je zložené z nasledujúcich polí:

- Blen – pole o veľkosti 12 bitov špecifikuje množstvo 8 bajtových alokácií v poli BWmap.
- Alen – toto pole je prítomné z historických dôvod, z doby kedy sa na prenos používal protokol ATM (Asynchronous Transfer Mode). Počas prenosu je vždy nastavené na hodnotu 0.
- CRC – kontrolný súčet, kontroluje a opravuje chyby.

Pole Plend je v zostupných rámcoch obsiahnuté dvakrát z dôvodu zvýšenia robustnosti prenosu. ONU teda vykoná dvojnásobné overenie CRC kódu na zistenie či BWmap a rámce sú použiteľné [14].

BWmap (Bandwidth Map)

BWmap špecifikuje priraďovanie prenosovej kapacity v zostupnom smere jednotlivým ONU. Obsahuje sériu 8 bajtových alokačných štruktúr, z ktorých každá garantuje povolenie na prenos jednotlivých dávok v zostupnom smere. Každá alokačná pozostáva z piatich polí ako je zobrazené na obrázku 2.6:



Obr. 2.6: Štruktúra BWmap

Krátky popis jednotlivých polí BWmap:

- Alloc—ID – tento identifikátor špecifikuje majiteľa vysielacieho grantu vo vzostupnom smere. Hodnoty menšie ako 254 priamo označujú samotné ONU a autorizujú prenos vo vzostupnom smere. Alloc—ID s hodnotou 254 iniciuje serial_number PLOAM odpoveď z neobjavených ONU. Alloc—ID s hodnotou 255 je rezervované [14].

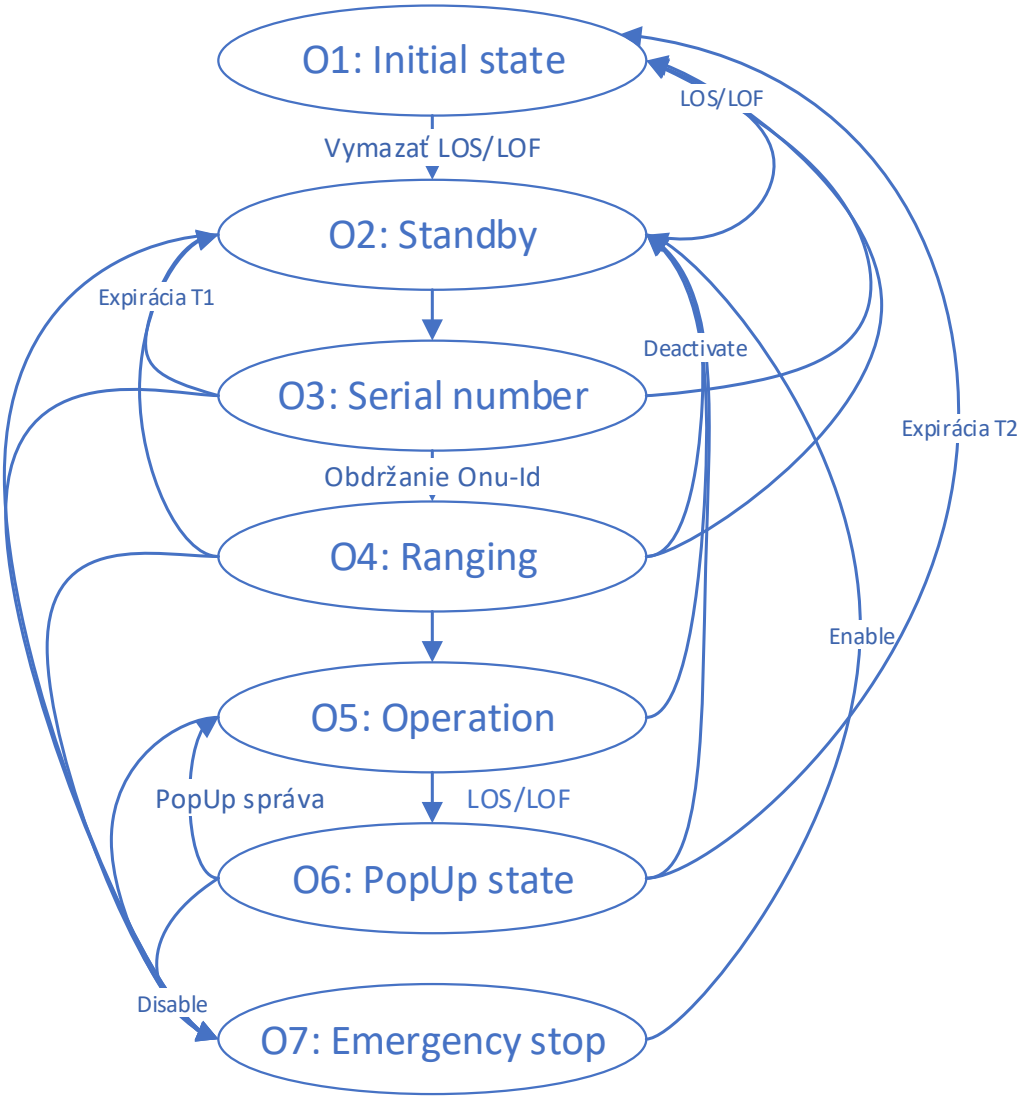
- Flags – toto pole má dĺžku 12 bitov z čoho 5 je využívaných. Toto pole určuje parametre prenosu vo vzostupnom smere [14, 15].
- Start Time – v tomto 16 bitovom poli je špecifikovaná doba medzi začiatkom alokačného štartovacieho času od začiatku rámca prenášaného vzostupným smerom.
- Stop Time – jedná sa taktiež o 16 bitové pole, ktoré určuje dobu od začiatku rámca.
- CRC (Cycle Redundancy Check) – každá alokačná štruktúre je chránená pomocou CRC-8, ktorý dokáže opraviť jednu chybu a spoľahlivo detekovať dve [15].

2.2.4 ONU Aktivačný proces

Každá ONU po pripojení, alebo znovu pripojení do siete PON, musí prejsť niekoľkými aktivačnými krokmi, ktoré nazývame aktivačný proces [14]. Aktivačný proces v sieťach pozostáva z troch fáz [13]:

1. Synchronization (parameter learning) – počas fáze parameter learning, ONU získava funkčné parametre, ktoré sú potrebné k vysielaniu vo vzostupnom smere.
2. Serial number acquisition (discovery) – počas fáze discovery je novo pripojená ONU objavená OLT stanicou na základe jej sériového čísla. Po objavení je stanici priradené unikátny identifikátor nazývaný ONU-ID.
3. Ranging

Behom svojho účinkovania v sieti PON, môže jednotka ONU nadobudnúť niekoľko operačných stavov [14]. Jednotlivé operačné stavy a prechody medzi nimi sú znázornené na obrázku 2.7.



Obr. 2.7: Aktivačný proces

O1: Initial state

V tomto stave dôjde k spusteniu samotnej ONU stanice. Po spustení stanica čaká na priatie signálu zo zostupného smeru. Po obdržaní signálu jednotka začne proces synchronizácie. Synchronizácia začína uvedením jednotky do tzv. stavu Huntstate, v ktorom jednotka vyhľadáva polia PSync (Physical Synchronization). Po nájdení PSync neobsahujúceho chyby, stanica nastaví čítač N na hodnotu jeden a prejde do predsynchronizačného stavu PreSync. Po prijatí bezchybného poľa PSync dôjde k inkrementácií čítača N, pri priati poľa PSync obsahujúceho chybu dôjde k navráteniu stanice do stavu Huntstate. Synchronizácia je úspešná po správnej detekcií M-1

po sebe nasledujúcich rámcov a prechádza do stavu Syncstate a začína spracovávať informácie z PCBd (Physical Control Block downstream) hlavičky. Pri detekcií nesprávneho rámca je jednotka prevedená do stavu Huntstate aj v prípade pokročilejšej fáze synchronizácie. Po ukončení procesu synchronizácia sú parametre LOS (Lost Of Signal) a LOF (Lost Of Frame) vymazané a stanica sa presunie z operačného stavu O1 do stavu O2 [13, 16].

O2: Standby state

Po úspešnom ukončení synchronizácie v zostupnom smere ostáva vykonať synchronizáciu v smere vzostupnom. ONU stanica nastaví svoje globálne parametre na základe priatej všesmerovej PLOAM správy typu Upstream_Overhead, bližšie popísanej v sekcií 2.3.1. Po úspešnom nastavení parametrov prejde stanica z operačného stavu O2 do O3 [13].

O3: Serial number state

Po nastavení globálnych parametrov OLT stanica zašle vše-smerovú správu so žiadosťou o odoslanie sériového čísla danej jednotky ONU. ONU stanica po obdržaní žiadosti odošle svoje sériové číslo príslušnej OLT. Aby počas odosielania sériového čísla nedošlo ku kolízií s normálnym prenosom je pred a po odoslaní žiadostí vytvorené Quiet Window je reprezentované GPON rámcom s prázdnym poľom BWmap trvajúcim 250 μs. Jednotka ONU na túto správu odpovedá PLOAM správou Serial_Number_ONU so sériovým číslom jednotky. Na základe sériového čísla dokáže jednotka OLT priradiť jednotke konkrétne ONU-ID. OLT stanica pomocou všesmerovej správy Assign_ONU_ID priradí jednotlivým ONU ich unikátne ONU-ID [16]. Po úspešnom nastavení ONU-ID sú ONU a OLT schopné medzi sebou komunikovať pomocou unicastových správ a jednotka sa presunie z operačného stavu O3 do stavu O4 [13, 14, 16].

O4: Ranging state

Vzostupný prenos z rôznych ONU musí byť synchronizovaný s hranicami GTC rámca. Aby sa vzdialenosti rôznych ONU od OLT javili ako rovnaké, je potrebné zaviesť vyrovnávacie oneskorenie pre každú ONU. Toto vyrovnávacie oneskorenie sa meria práve, keď je jednotka v stave O4. Akonáhle stanice obdrží PLOAM správu typu Ranging_Time, popísanú v sekcií 2.3.1, presunie sa do stavu O5 [16].

O5: Operation state

V tomto stave môže ONU stanica plnohodnotne odosielať dáta a PLOAM správy na základe inštrukcií z OLT [13].

O6: PopUp state

ONU stanica sa prepne do tohoto operačného stavu po detekcií LOS alebo LOF alarm počas bežného používania v operačnom stave O5. Po prepnutí do PopUp stavu, jednotka okamžite zamedzí vysielanie signálu vzostupným smerom, čoho dôsledkom nadradená OLT stanica detekuje LOS alarm od danej jednotky. Jednotka v tomto stave spustí časový interval T2 počas, ktorého sa pokúša o opätované nadviazanie komunikácie a synchronizáciu. Ak sa jednotke nepodarí úspešne obdržať cielenú, alebo vše-smerovú správu a teda sa ani nevráti priamo do operačného stavu (O5) alebo do stavu ranging (O4), jednotka pravdepodobne nedokáže obnoviť optický signál a synchronizáciu GTC rámca. V takomto prípade sa jednotka po vypršaní časového intervalu T2 vráti do stavu O1 [16].

O7: Emergency stop state

Stanica, ktorá prijme PLOAM správu typu Disable_Serial_Number 2.3.1 správu s parametrom „disable“ prepne sa do stavu Emergency stop a vypne vysielací laser. Po odstránení dôvodu prerušenia činnosti danej stanice, je ONU stanica znovu aktivovaná odoslaním PLOAM správy Disable_Serial_Number s parametrom „enable“ z OLT stanice.

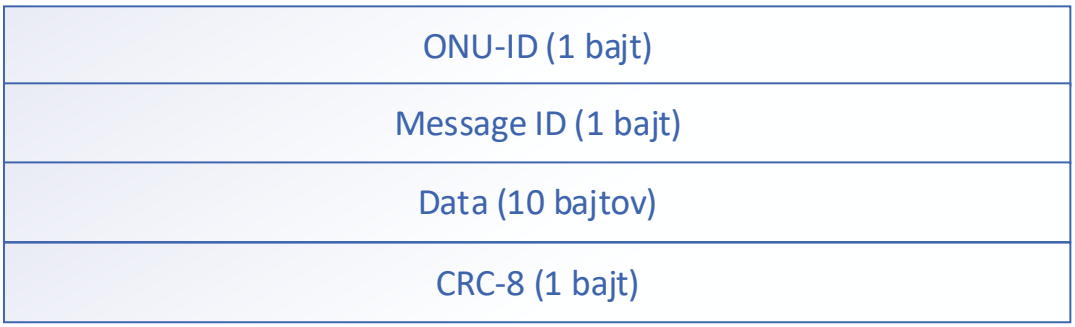
2.3 PLOAM

PLOAM kanál obsahuje sadu jednoduchých správ, ktoré sú podporované pomocou relatívne jednoduchého firmvéru, alebo aj samotným hardvérom. Používajú sa na dojednanie parametrov, signalizáciu a objavovanie na nízkej úrovne medzi jednotkou ONU a OLT stanicou, vrátane objavovanie nových ONU staníc v prípade, že nie je k dispozícií žiaden riadiaci kanál na vyššej vrstve.

2.3.1 Typy PLOAM správ v sieťach GPON

GPON PLOAM správy majú všetky dĺžku 13 bajtov a identickú štruktúru, ktorá je zobrazená na obrázku 2.8:

V zostupnom smere sú hodnoty 254 a 255 poľa ONU-ID rezervované. Všesmerové vysielanie má rezervovanú hodnotu 255 a hodnota 254 je využívaná ako alloc-ID pre nezaregistrované ONU jednotky [14]. Nižšie sú popísané niektoré z rôznych typov

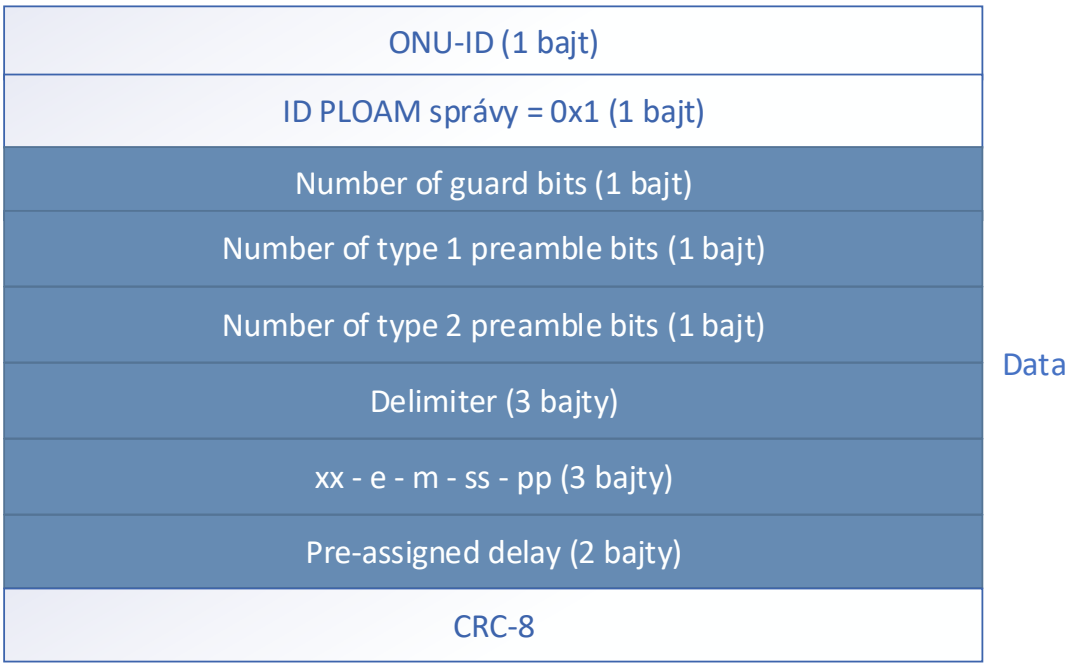


Obr. 2.8: Štruktúra PLOAM správy

PLOAM správ typických pre zostupný smer v sieťach G-PON. Jednotlivé typy správ sa odlišujú unikátnym Message-Id.

Upstream_overhead Message

G-PON ONU stanica nemá povolené vysielať, kým neprijme PLOAM správu typu upstream_overhead 2.9. ID tejto PLOAM správy má decimálnu hodnotu 1.



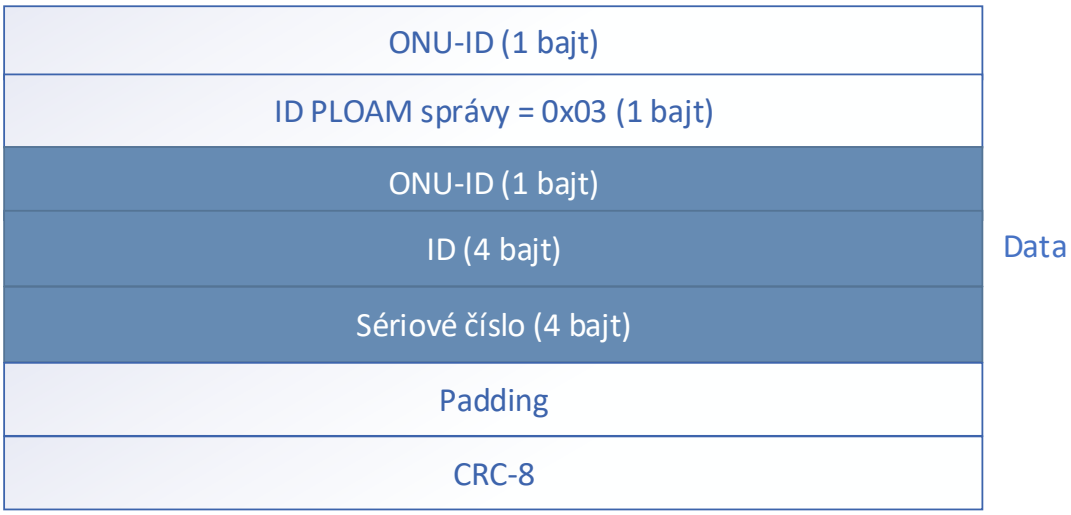
Obr. 2.9: Upstream_overhead Message

Bližšie informácie o jednotlivých poliach:

- Delimiter – pole dĺžky 24 bitov, ktoré obsahuje informácie o burst hlaivčke
- xx – rezervované bity
- e – ak je bit e nastavený na hodnotu 0, jednotka ONU začne ignorovať predNastavené oneskorenie
- m – tento bit je stále nastavený na hodnotu 0
- ss – toto pole vždy nastavené na hodnotu 00
- pp – toto pole nastaví jednotke ONU jeden z troch možných výkonnostných stavov, avšak v sieťach typu G-PON táto funkcionalite nie je využívaná.

Assign_ONU-ID Message

Priradenie ONU-ID novo objavenej jednotke znamená prechod jednotky zo stavu serial_number (O3) do ranging stavu (O4). Akonáhle ONU opustí stav O3, jej ONU-ID nemôže byť zmenené až kým nebude jednotka deaktivovaná. Štruktúra správy ne vyobrazená na obrázku 2.11



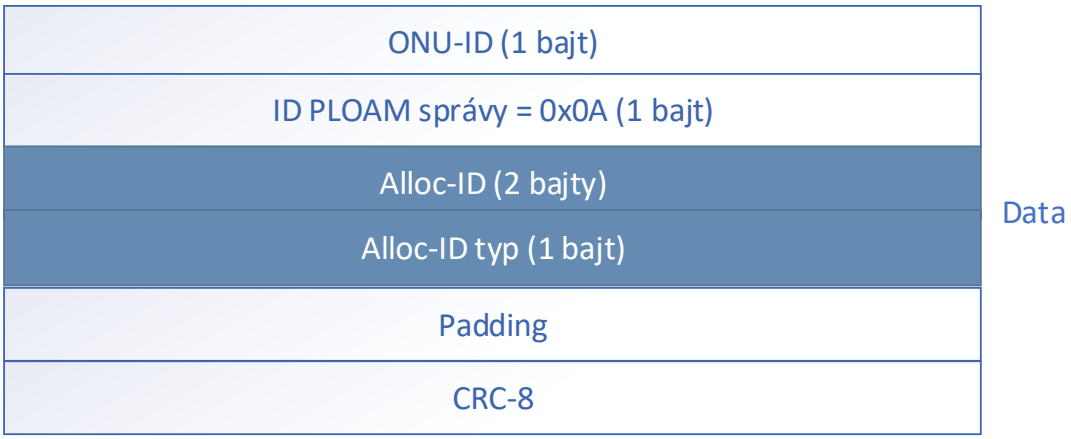
Obr. 2.10: Assign_ONU-ID Message

Ranging_time Message

Pomocou tejto správy stanica OLT nastaví vyrovnávajúce oneskorenie (equalization delay). Táto správa sa využíva pri aktivačnom procese, ale aj vždy keď je to potrebné počas operovania ONU jednotky [14].

Assign_alloc-ID Message

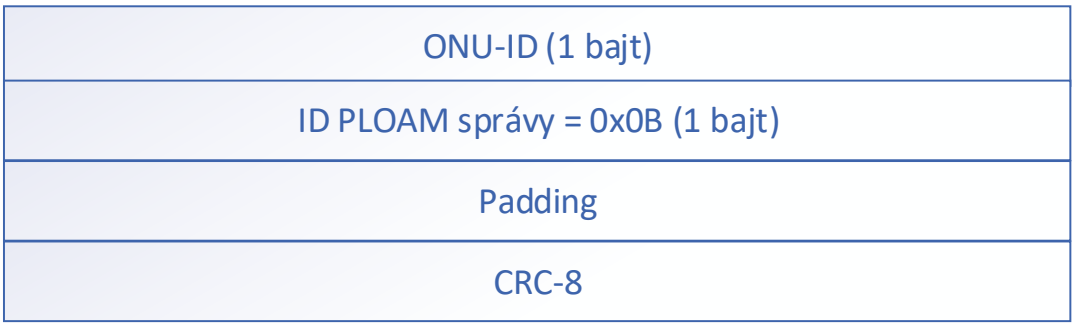
Základná hodnota alloc-ID danej ONU je rovná ONU-ID a nemôže byť zmenená. Pole alloc-ID má dĺžku 12 bitov a je rozdelené do dvoch bajtov, kde posledné 4 bity sú vždy rovné 0 [14].



Obr. 2.11: Assign_alloc-ID Message

No_message Message

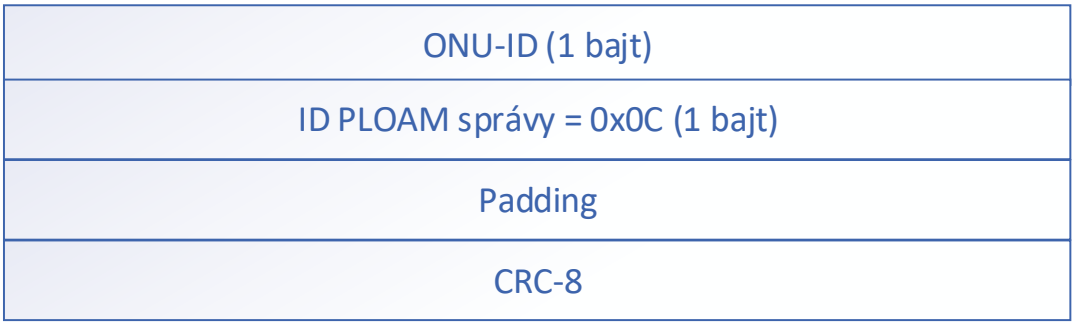
Štruktúra GPON rámcov je definovaná tak, že OLT stanica odosiela PLOAM správu v každom rámci bez podmienok. Ak OLT stanica nemá žiadne dáta k odoslaniu, odošle správu typu No_message, ktorej Message_id je 11. Štruktúra správy je zobrazená na obrázku 2.12.



Obr. 2.12: No_message Message

Popup Message

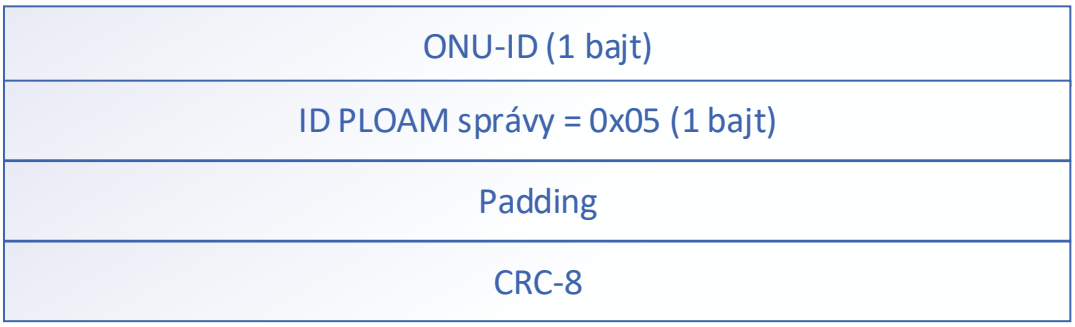
V sekcií 2.2.4 sme si opísali, že ak na G-PON ONU dôjde ku strate zostupného signálu alebo k desynchronizácií, jednotka sa uvedie do tzv. PopUp stavu (O6). Ak sa ONU jednotke podarí obnoviť svoju činnosť predtým ako vyprší časovač, alebo ak sa stanici OLT podarí dostatočne rýchlo detekovať výpadok spojenia a obnoviť ho, PopUp správa 2.13 môže ONU prepnúť znova do operačného stavu (O5) bez procesu opätovnej synchronizácie [14]. Tento typ správy má aj všesmerovú verziu, ktorá spôsobí že všetky stanice nachádzajúce sa v PopUp stave (O6) sa prepnú do stavu Ranging state (O4). ID tohoto typu správy má v decimálnom tvare hodnotu 12.



Obr. 2.13: PopUp Message

Deactivate_ONU-ID Message

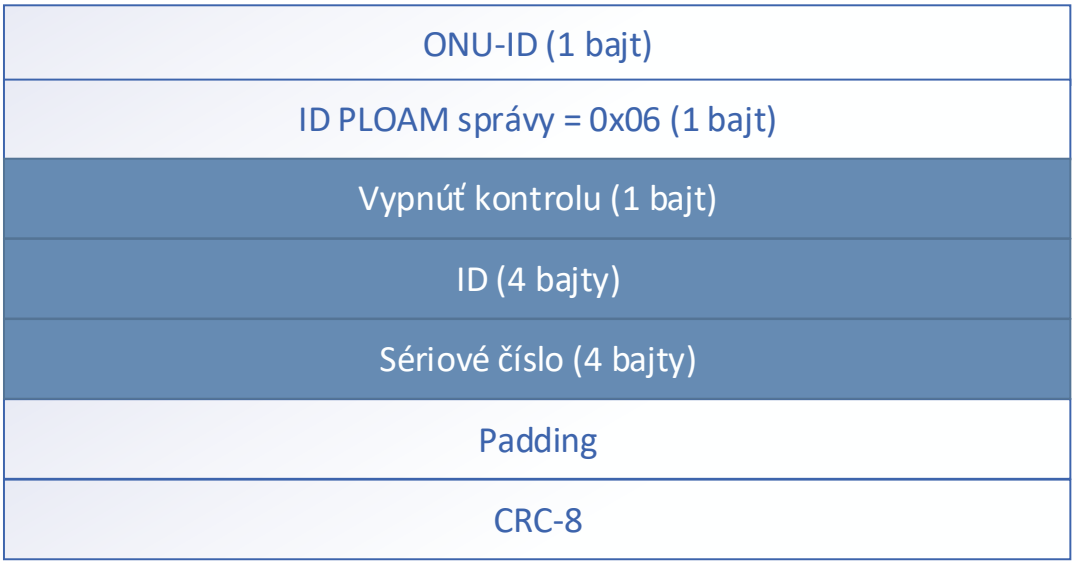
Po prijatí tejto správy 2.14 je jednotka ONU inštruovaná k tomu aby odstránila všetky svoje nadobudnuté údaje poskytnuté TC vrstvou a presunula sa do stavu Standby (O2). V tomto stave jednotka dokáže odpovedať na správy prideľujúce sériové číslo, je viditeľná pre OLT stanicu a je reaktivovaná do siete PON. ID tohoto typu správy má v decimálnom tvare hodnotu 5.



Obr. 2.14: Deactivate_ONU-ID Message

Disable_serial_number Message

Touto správou 2.15 sa OLT stanica pokúša odstaviť ONU jednotku ešte predtým ako jej neobvyklé správanie môže spôsobiť škody. ID tohoto typu správy má v desiatkovej sústave hodnotu 6. To je však možné len za predpokladu, že ONU jednotka ešte dokáže rozpoznať tento typ správy a kontrolovať svoj vysielač.



Obr. 2.15: Disable_serial_number Message

Pole vypnutia kontroly (disable control) môže nadobúdať hodnoty:

- 0xFF – jednotke ONU bol odopretý prístup na vysielanie vo vzostupnom (upstream) smere a zároveň bola prepnutá do stavu Emergency stop (O7) 2.2.4.
- 0x0F – všetky jednotky, ktoré sa nachádzajú v stave O7 a obdržia túto správu, sú prepnuté do stavu Standby (O2) v ktorom môžu byť znova objavené OLT

stanicou a aktivované ako nové jednotky do siete PON. Polia ID (Vendor ID) a Sériové číslo (Vendor-specific serial number) sú v tomto prípade ignorované [13].

- 0x00 – ONU stanica so špecifickým ID a sériovým číslom, je prepnutá do stavu Standby (O2), kde môže byť znovu objavená OLT stanicou a reaktivovaná do siete PON. Ak sa daná ONU nenachádza v stave Emergency state (O7) tak túto správu ignoruje.

3 Návrh a implementácia systému

V tejto kapitole je popísaný spôsob akým bude Apache Kafka využitý v systéme na monitorovanie pasívnych optických sietí a spôsob akým budú rámce ďalej spracovávané, filtrované a následne uložené pre ďalšie spracovanie. Podobne v tejto kapitole budú priblížené samotné komponenty systému, spôsob akým fungujú a samotná implementácia týchto komponentov primárne s využitím technológií Python a Docker. Všetky zdrojové kódy spolu s informáciami v podobe vygenerovanej dokumentácie sú dostupné v elektronickej prílohe tejto práce.

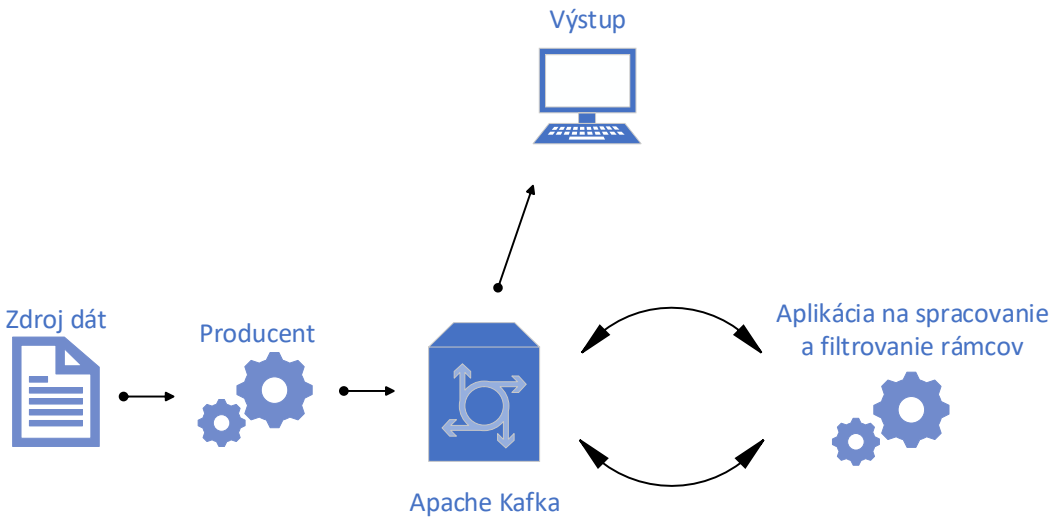
Príklady implementácie sú sprevádzané fragmentami použitého kódu a najdôležitejších využívaných príkazov. Ako bolo spomenuté vyššie aby bolo zabezpečená platformná nezávislosť systému, sú všetky jeho komponenty implementované ako nezávislé docker kontajnery, ktoré medzi sebou spolupracujú. K implementácií klientov ako producent a konzument nemusí byť nutne využitý len jazyk Python. Implementácie týchto komponentov exitujú v podstate vo všetkých existujúcich programovacích jazykoch a na ich používanie stačí v rámci skriptu importovať správne knižnice. V závere kapitoly je popísaný výstup spomínaného systému, formát uložených a spracovaných informácií získaných z GPON rámcov a predstavené spôsoby ako tieto dáta získať a interpretovať.

Základný pohľad na systém a jeho primárne komponenty, je zobrazený na obrázku 3.1. Bližší popis samotných komponentov, ich funkcionalít a ich implementácia sa nachádza v sekciách 3.3, 3.4 resp. 3.5. Spoločným znakom týchto komponentov je primárne to, že sú tvorené samostatným docker kontajnerom, preto v sekcií 3.1.1 nájdeme popis a príklady implementácie jednotlivých docker kontajnerov tvoriacich samotné komponenty ako aj priblíženie systému Docker ako takého.

V rámci diplomovej práce bol vypracovaný návrh uloženia a prenosu rámcov prostredníctvo platformy Apache Kafka. Prostredníctvom aplikácie napísanej v jazyku Python sú spracovávané rámce filtrované, triedené a sú z nich extrahované informácie relevantné pre monitorovanie GPON siete a udalostí v nej. Takto spracované rámce a informácie z nich získané, sú uložené v Apache Kafka pripravené na ďalšie spracovanie. Rámce, ktoré sú k dispozícií na spracovanie sú zachytené z komunikácie v zostupnom smere.

3.1 Docker

Keďže jednotlivé komponenty navrhnutého systému na sledovanie udalostí v sieťach GPON sú postavené na podstate Docker kontajnerov, v úvode tejto kapitoly je v krátkosti priblížený samotný Docker a jeho význam vo svete softvérového vývoja.



Obr. 3.1: Architektúra systému

Taktiež vrámci tejto sekcie je predstavená základná architektúra tohoto systému, jednotlivé kontajnery, ich približný obsah a vzájomná komunikácia.

Docker je jedna z popredných otvorených platforiem na vývoj a kontajnerizáciu aplikácií. Poskytuje spôsob ako zabaliť infraštruktúru aplikácie do kontajnera, kde môže bežať izolovane od zvyšku systému. Docker zjednodušuje vývojový proces a to umožnením rozdelenia rozsiahlejších projektov na jednotlivé mikroslužny bežiace nezávisle na sebe v kontajnerovom prostredí [17].

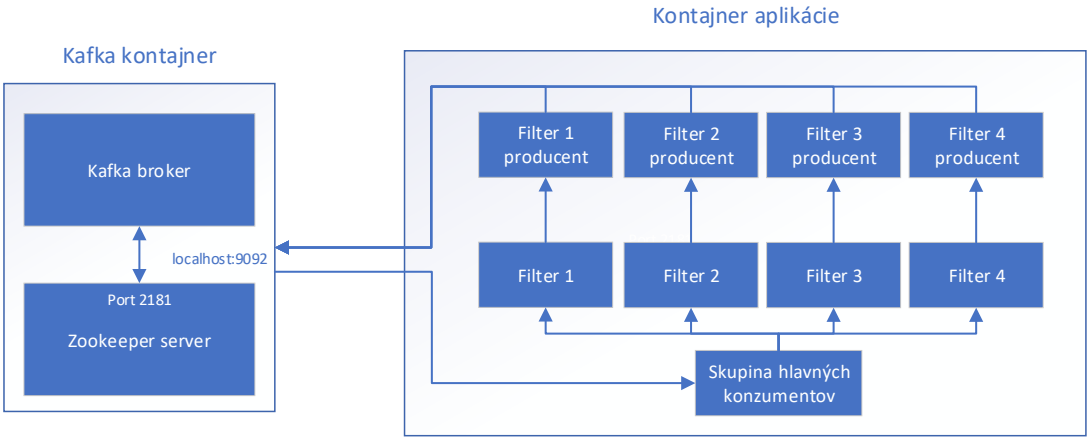
3.1.1 Používanie nástroja Docker

Pri práci s nástrojom Docker dochádza primárne k vytváraniu dvoch rôznych objektov a to obrazov (images) a kontajnerov (containers).Obrazy sú read-only šablóny s príkazmi slúžiacimi na vytvorenie Docker kontajnera. Na vytvorenie obrazu stačí vytvoriť súbor Dockerfile, ktorý obsahuje dané príkazy. Zavolaním daemona sa tieto príkazy vykonajú krok po kroku, kde každý krok vytvára novú vrstvu a na konci vytvoria kompletný obraz. V prípade, že sa zmení nejaký príkaz, stačí ak docker vytvorí iba novú vrstvu, vďaka čomu sú obrazy malé a rýchle. Obrazy môžu byť aj založené na iných obrazoch, čo nám dovoľuje zobrať už vytvorený obraz a iba ku nemu pridať našu funkcionalitu [17].

Kontajner je fungujúca inštancia obrazu. Je vytvorený na základe príkazov použitých na vytvorenie obrazu, alebo príkazov, ktorými pridávame možností pri jeho zapnutí [17, 18].

3.1.2 Docker v systéme na monitorovanie udalostí v sieťach GPON

Systém vytvorený vrámci tejto diplomovej práce pozostáva z dvoch docker kontajnerov 3.2. Kontajner s názvom Kafka kontajner je zložený z dvoch základných obrazov a to zo samotného Kafka brokera a zo Zookeeper servera, ktorý Kafka k svojej funkčnosti nevyhnutne potrebuje. Detailnejší popis tohoto kontajnera a jeho obrazov je popísaný v sekcií 3.4.



Obr. 3.2: Docker architektúra systému

Kontajner aplikácie, ktorá zabezpečuje filtrovanie a ďalšie spracovávanie GPON rámcov je taktiež zobrazený na obrázku 3.2. Tento kontajner tvorí primárne aplikácia napísaná v jazyku Python. Táto aplikácie je tvorená skupinou hlavných konzumentov, ktorých úlohou je paralelne odoberať rámce z Kafka témy GPONFrames. Následne sa na rámcoch a ich dátach vykonávajú rôzne operácie spracovávania a filtrovania. Po tom čo sú rámce spracované, informácie získané z nich sú pomocou producentov odosielané do príslušných tém v Kafka kontajnery, kde sú pripravené na ďalšie spracovanie.

3.2 Formát prenášaných GPON rámcov

Súbor vstupných dát pozostáva z viacerých textových súborov, ktoré obsahujú istú množinu GPON rámcov vo formáte JSON. V prílohe A.1 je uvedený príklad jedného zo spracovávaných rámcov. Štruktúra rámcov je v podobe kľúč – hodnota kde kľúč predstavuje jednotlivé bunky tvoriace GPON rámec v zostupnom smere, tak ako je bližšie popísané v kapitole 2.2.3.

3.3 Producent

V tejto sekcií je bližšie popísaný producent ako komponenta architektúry vyobrazenej na obrázku 3.1. V tomto prípade komponent, ktorý zabezpečuje napĺňanie Apache Kafka GPON rámcami určenými na ďalšie spracovanie, nie je tvorený docker kontajnerom. Formát rámcov je bližšie špecifikovaný v sekcií 3.2. Dôvodom prečo nie je tento komponent „dockerizovaný“ je ten, že tento komponent bude s vysokou pravdepodobnosťou použitý priamo v zdrojových kódoch zariadenia, ktoré samotné GPON rámce odchytáva a producent popísaný nižšie slúži ako príklad. V rámci diplomovej práce je tento komponent tvorený skriptom napísaným v jazyku Python a je zložený z dvoch častí. Prvá časť zabezpečuje manipuláciu s rozsiahlými zdrojovými súbormi (vstupná množina testovacích dát) a extrahovaním jednotlivých rámcov z nich. Druhou časťou je samotný Apache Kafka klient ktorý zabezpečuje odosielanie extrahovaných rámcov do konkrétnej témy.

K vytvoreniu producentov v jazyku Python bol využitý modul *kafka-python* a jeho trieda *KafkaProducent*. Na to aby sme tento modul vedeli používať, je nutné ho do prostredia, kde bude tento skript spúšťaný nainštalovať. Inštaláciu modulu môžeme spustiť napríklad pomocou príkazu 3.5:

Výpis 3.1: Inštalácia modulu kafka-python

```
1 >. pip install kafka-python
```

Kľúčovým prvkom programu je vytvorenie inštancie triedy KafkaProducent a jeho konfigurácia tak ako je vyobrazené na príklade 3.2.

Výpis 3.2: Vytvorenie producenta

```
1 producer = KafkaProducer(bootstrap_servers=['localhost:9092'],
2                value_serializer=lambda v: json.dumps(v).encode('utf-8'),
3                api_version=(0, 10, 1))
```

Dôležitými parametrami KafkaProducer konštruktora sú:

- bootstrap_servers – IP adresa a číslo portu na ktorom je spustený server Apache Kafka. V tomto prípade sa jedná o IP adresu lokálneho PC a port 9092, čo je východzie číslo portu Kafka serveru.
- value_serializer – metóda na serializáciu dát pred odoslaním na server. V tomto prípade ide o lambda metódu, ktorá zabezpečí, že dáta budú pred odoslaním vo formáte JSON.

- api_version – verzia API, ak nie je určená explicitne, producent skúša rôzne verzie kým nenarazí na verziu podporovanú serverom.

Súbory použité ako zdroje dát boli umiestnené v adresári s názvom *gpon_frames*. Na to aby skript pracoval so všetkým zdrojovými súbormi v tomto adresári, bol využitý modul *glob*. Program postupne iteruje všetkými súbormi s príponou *„.txt“* a ich obsah parsuje na odosielateľné rámce vo formáte JSON. Samotné odosielanie prebieha v rámci iterácie vykonštruovaného *response* objektu, ktorý obsahuje jednotlivé GPON rámce, tak ako je vyobrazené na príklade 3.4.

Pred samotným odosielaním dát, je ešte potrebné vytvoriť cieľovej téme niekoľko partícií. Partície sú potrebné preto, aby bolo možné využiť úplný a odporúčaný potenciál, ktorý ponúka možnosť vytvorenia skupiny konzumentov (consumers group) [19]. V prípade vytvorenia skupiny konzumentov, sú dáta z partícií ekvivalentne prideľované jednotlivým konzumentom, ktorý sú do tejto skupiny pripojený, čím je možné dosiahnuť rozprestrenie záťaže (load ballancing) a samozrejme aj paralelné spracovávanie rámcov. Samotná implementácia skupiny konzumentov je popísaná v texte nižšie. Napríklad v prípade vytvorenia skupiny konzumentov s desiatimi členmi, ktorý konzumujú jednu tému, by v ideálnom prípade mala daná téma obsahovať desať partícií. Automatický „load ballancing“ rozdelí zátaž tak, že každému z konzumentov priradí práve jednu partíciu, čím je spracovávanie správ ideálne vyvážené. K automatickému rozloženiu zátaže skupine konzumentov dôjde za predpokladu, že inštancie konzumentov využívajú metódu *subscribe()* [21] V tomto prípade vytvoríme partície dve, pretože na strane aplikácie bude skupina konzumentov obsahovať dvoch členov, samozrejme je možné tento počet zvýšiť no pre obavu zo zložitosti manažmentu veľkého počtu vláken (thread management) na strane aplikácie, bol vytvorený model s dvoma vláknami resp. dvoma konzumentmi. Na výpise 3.16 je uvedený príklad manuálneho priradenia témy a špecifickej partície inštancií konzumenta, pomocou metódy *asign()* [21].

Partície môžeme vytvoriť pomocou metódy zobrazenej na výpise 3.3. Ku konfigurácií Kafka servera vytvoríme inštanciu *KafkaAdminClient*. Následne pomocou metódy *create_partitions()* vytvoríme daný počet partícií pre tému GPONFrames.

Výpis 3.3: Vytvorenie partícií

```
1 def create_partitions():
2     admin_client = KafkaAdminClient(bootstrap_servers=['localhost:9092'])
3     topic_partitions = {}
4     topic_partitions['GPONFrames'] = NewPartitions(total_count=2)
5     admin_client.create_partitions(topic_partitions)
```

Odoslanie správy do Kafky prebieha pomocou metódy *send()*. Táto metóda je prednastavená v asynchrónnom móde. Po jej zavolaní, je daná správa pridaná do vyrovnávacej pamäte, kde spolu s ostatnými správami čaká na odoslanie. To umožňuje producentovi odosielať záznamy v dávkach, čo maximalizuje efektivitu odosielania [20]. V prípade, že odosielame správy do témy, ktorá ešte nebola na Kafka servery definovaná, bude téma s týmto názvom vytvorená automaticky.

Producent udržiava záznamy neodoslaných záznamov pre každú partíciu v cieľovej téme. Veľkosť vyrovnávacej pamäte jednotlivých partícií je určená parametrom *batch_size*, ktorý je nepovinným parametrom v konštruktore producenta. Názov témy do ktorej má byť správa publikovaná je jediným povinným parametrom tejto metódy. Medzi ďalšie parametre patria: hodnota resp. dáta na odoslanie (value), cieľová partícia (partition), kľúč ktorý chceme asociovať s danou správou a časová známka (timestamp) [20]. V prípade, že téma obsahuje viacero partícií, môžeme v metóde *send()* špecifikovať číslo partície, do ktorej chceme správy publikovať. V prípade, že tento parameter nenastavíme, producent bude publikovať správy podľa prednastavených nastavení. Na príklade môžeme vidieť producenta produkujúceho dáta do témy s názvom *GPONFrames* a to partície *0* 3.4.

Celý zdrojový kód producenta je k dispozícií v prílohe číslo B.1.

Výpis 3.4: Odosielanie rámcov

```
# Iterate through all files from folder
for file in files:
    with open(file, 'r') as f:
        response = json.dumps(f.read())
        response = ast.literal_eval(json.loads(response))

        # Iterate through JSON objects (frames)
        for i in range(1, len(response)):
            data = {i: response[i]}
            producer.send('GPONFrames', value=data, partition=0)
```

3.4 Apache Kafka

V rámci tejto sekcie je popísaný postup implementácie Apache Kafka komponentu, ktorý je tvorený docker kontajnerom. Kontajner je tvorený obrazmi Kafka brokera a Zookeeper servera. Kafka broker obsahuje niekoľko tém v ktorých sú držané samotné GPON rámce a v niekoľkých témach sú uložené získané informácie z rámcov.

3.4.1 Tvorba Zookeeper kontajnera

Zookeeper je centralizovaná služba spoločnosti Apache a používa sa na identifikáciu uzlov, uchovávanie konfiguračných údajov a na zabezpečenie robustnej a flexibilnej konfigurácie medzi distribuovanými systémami. Udržuje informácie a sústavne monitoruje jednotlivé uzly v rámci daného Kafka clusteru, sleduje ich témy a partície. Zookeeper umožňuje simultánne zapisovanie a čítanie viacerých klientov a správa sa ako zdieľaný konfiguračný servis v danom systéme. Avšak v najbližšej dobe sa očakáva odstránenie závislosti Kafky od tohoto softvéru [9]. Tento softvér je súčasťou softvéru Apache Kafka. Ich zdrojové súbory sú dostupné na stiahnutie z oficiálnych Kafka stránok na odkaze *https://kafka.apache.org/downloads*.

Zookeeper server musí byť spustený ako prvý, teda pred samotným spustením Kafka brokera. Pred samotným spustením je ešte potrebné nakonfigurovať cestu, kde má server ukladať dáta. Konfiguračný súbor sa nachádza v adresári, kde boli extrahované zdrojové súbory Kafky a to na nasledujúcej adrese *.\config\zookeeper.properties*. Cestu nastavíme editovaním parametru dataDir a volíme ideálne cestu do adresára s extrahovanými konfiguračnými súbormi.

Výpis 3.5: Inštalácia modulu kafka-python

```
1 >. pip install kafka-python
```

Ako bolo spomenuté v sekcií 3.1.1, docker obraz vytvoríme pomocou Dockerfile súboru. Príklad obsahu súboru, potrebného k vytvoreniu obrazu Zookeeper server je na výpise 3.6. Názov súboru nastavíme kvôli lepšej organizácií a prehľadnosti na *Dockerfile_zookeeper*.

Výpis 3.6: Zookeeper Dockerfile

```
1 # Dockerfile for Zookeeper server
2
3 # Get base ubuntu image
4 FROM ubuntu:latest
```

```
5
6  MAINTAINER Matej Pancak
7
8  EXPOSE 2181
9
10 CMD ["bin/zookeeper-server-start.sh", "config/zookeeper.properties"]
11
12 WORKDIR /kafka
13
14 COPY src/kafka_2.13-2.7.0 .
15
16 RUN chmod +x . -R
17 RUN apt update
18 RUN apt install -y openjdk-8-jdk
19 RUN mkdir /kafka-data
```

Ako základný obraz, na ktorý postupne budeme pridávať ďalšie vrstvy tohoto obrazu, je použitá posledná stabilná verzia Ubuntu, tento parameter nastavíme pomocou príkazu *FROM* nasledovaný menom obrazu, v tomto prípade *ubuntu:latest*. Ak sa tento základový obraz aktuálne nenachádza v systéme, docker ho počas vytvárania obrázku získa z knižnice obrazov nazývanej Dockerhub. Príkazom *EXPOSE* určíme port na ktorom bude následne vytvorený kontajner naslúchať, v tomto prípade hodnotu nastavíme na predvolenú hodnotu Zookeeper servera teda *2181*. Pomocou príkazu *CMD* definujeme príkaz, ktorý sa spustí bezprostredne po spustení zbuildovaného obrazu, teda vytvorení kontajnera. V tomto prípade sa jedná o príkaz slúžiaci na spustenie Zookeeper servera. Do hranatých zátvoriek postupne vyplníme na prvú pozíciu príkaz, ktorý sa má vykonať a na druhú povinný argument, v tomto prípade cestu ku konfiguračnému súboru 3.7.

Výpis 3.7: Spustenie Zookeeper servera

```
1 CMD ["bin/zookeeper-server-start.sh", "config/zookeeper.properties"]
```

Ďalšími príkazmi tvoriacimi docker obraz sú *WORKDIR*, pomocou ktorého sa presunieme do pracovnej zložky a *COPY*, ktorý do súčasnej zložku nakopíruje zdrojové súbory Kafky, ktorých získavanie bolo popísané v texte vyššie. Nasledujúcou sériou príkazov *RUN* dôjde k inštalácií Javy, ktorá je pre funkčnosť Kafky a Zookeeper servera nevyhnutná.

3.4.2 Tvorba docker kontajnera Apache Kafka

Docker obrazu pre Apache Kafka brokera bol vytvorený obdobným spôsobom ako tomu bolo v prípade Zookeeper servera 3.9. Ako základný obraz je opäť použitá posledná stabilná verzia Ubuntu. Porty, na ktorých bude daný kontajner sú v tomto prípade 9092 a 9093. Príkaz, ktorý bude v tomto prípade vykonaný bezprostredne po spustení kontajnera je zobrazený na výpise 3.8.

Výpis 3.8: Spustenie Kafka servera

```
CMD ["bin/kafka-server-start.sh", "config/server.properties"]
```

V rámci tvorby obrazu sú taktiež ako v prípade Zookeepera nakopírované zdrojové súbory Kafky do priečinka */kafka* a nainštalovaná Java.

Výpis 3.9: Kafka Dockerfile

```
# DockerFile for Apache Kafka

# Get base ubuntu image
FROM ubuntu:latest

MAINTAINER Matej Pancak <xpanca00@vutbr.cz>

EXPOSE 9092 9093

CMD ["bin/kafka-server-start.sh", "config/server.properties"]

WORKDIR /kafka

COPY src/kafka_2.13-2.7.0 .

RUN chmod +x . -R
RUN apt update
RUN apt install -y openjdk-8-jdk
RUN mkdir /kafka-data
```

Jednotlivé témy pre Apache Kafka nie je potrebné manuálne vytvárať vopred. V prednastavenom režime v prípade že producent produkuje správy do témy, ktorá neexistuje, je táto téma automaticky vytvorená.

3.4.3 Spustenie kontajnerov

V tomto bode práce je popísané ako vyššie spomenuté kontajnery tvoriace komponentu s názvom Apache Kafka spustiť. Spúšťanie a manažment aplikácií tvorenými viacerými docker kontajnermi nám umožňuje nástroj s názvom *docker-compose*. Tento nástroj je súčasťou balíka docker, teda ak na danom systéme je nainštalovaný docker, s najvyššou pravdepodobnosťou bude tento systém disponovať aj nástrojom docker-compose. Tento nástroj využíva zdrojové súbory vo formáte *.YAML*, podobne ako to bolo pri definovaní skladby samotných docker obrazov. Pomocou tohoto súboru vieme, spustiť viacero kontajnerov naraz, s rôznymi konfiguráciami. Na výpise 3.10 je príklad docker-compose súboru, ktorý zabezpečí spustenie a vykonanie základných príkazov na kontajneroch Kafky a Zookeeper servera.

Výpis 3.10: Príklad docker-compose súboru

```
# docker-compose file for Apache Kafka component

version: "3.4"
services:
#Build an run Apache Kafka service
  kafka:
    build:
      context: .
      dockerfile: Dockerfile_kafka
    image: kafka
    network_mode: host
    container_name: kafka
    depends_on:
      - zookeeper

#Build and run Zookeeper service
  zookeeper:
    build:
      context: .
      dockerfile: Dockerfile_zookeeper
    image: zookeeper
    container_name: zookeeper
    network_mode: host
```

Každý docker-compose súbor sa musí začínať špecifikovaním verzie súboru pomocou parametru *version*. V tomto prípade je verzia súboru 3.4. Následne príka-

zom *services* špecifikujeme jednotlivé služby (kontajnery), ktoré budú vrámci tohoto súboru spúšťané. Ako prvá služba je definovaná Kafka. V parametri *build* špecifikujeme cestu a názov Dockerfile súboru z ktorého bude daný kontajner vytvorený. Príkazom *network_mode* zabezpečíme aby daný kontajner bol dostupný aj priamo z lokálnej siete vrámci stroja. Toto nastavenie uľahčí konfiguráciu vzájomnej komunikácie jednotlivých komponentov systému vytvoreného vrámci tejto práce. Teda v tomto prípade bude kafka kontajner dostupný na adrese localhost:9092. Príkazom *depends_on* definujeme poradie spustenia služieb. V tomto prípade sa služba kafky spustí až po tom, čo bude spustená služba zookeepera.

Ako druhú službu definujeme službu Zookeeper servera. Obdobne ako v predchádzajúcom príklade špecifikujeme Dockerfile k príslušnému obrazu, a nastavíme parameter *network_mode*.

Následné spustenie týchto kontajnerov pomocou nástroja docker-compose vykonáme pomocou príkazu nižšie 3.11. Tento príkaz spustíme z terminálového okna otvoreného v priečinku, kde sa nachádza docker-compose.yml súbor.

Výpis 3.11: Spustenie docker-compose

```
1 >. docker-compose up -d
```

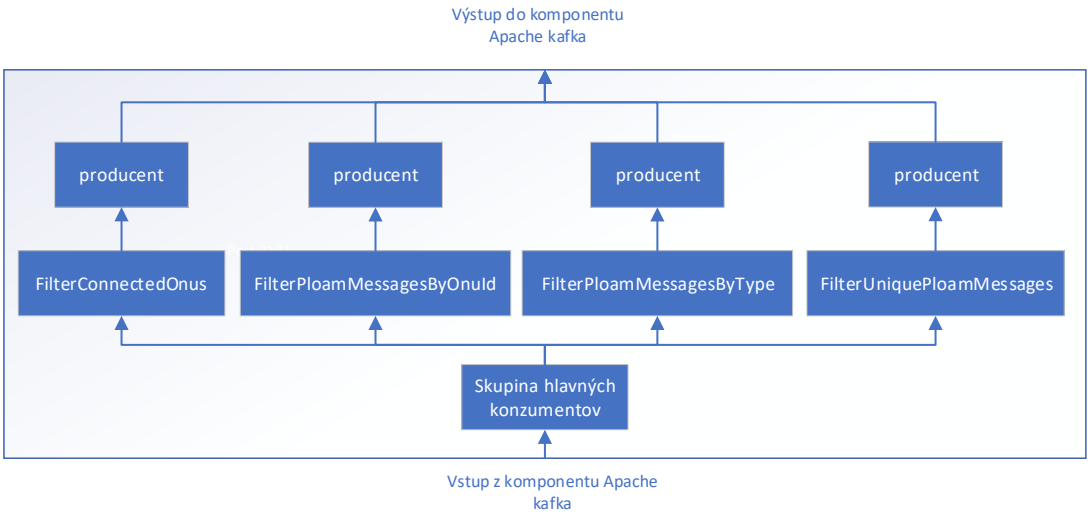
Argument *-d* znamená že daná kontajneri budú spustené na pozadí, nie v danom terminálovom okne. Zoznam a informácie o spustených kontajneroch zobrazíme príkazom 3.12.

Výpis 3.12: Zobrazenie informácií o spustených kontajneroch

```
1 >. docker ps
```

3.5 Aplikácia

V rámci tejto sekcie je popísaná štruktúra a jednotlivé komponenty tvoriace aplikáciu na spracovávanie a filtrovanie GPON rámcov. Zdrojové kódy metód a tried spomínaných v texte nižšie sú dostupné v priloženom dátovom úložisku spolu s krátkou dokumentáciou. V práci sú uvádzané len krátke úryvky kódu a to z dôvodu rozsiahlosti vytvoreného kódu. Táto aplikácia napísaná v jazyku Python obsahuje niekoľko konzumentov a producentov a je tvorená niekoľkými filtrami, ktoré zabezpečujú získavanie potrebných informácií z GPON rámcov. Táto aplikácia je taktiež vyhotovená do podoby docker kontajnera, čo umožňuje jej bezproblémovú funkcionalitu naprieč rôznymi platformami a systémami. Jedinou podmienkou je mať nainštalovaný a funkčný docker a docker-compose. Základné komponenty a architektúra sú vyobrazené na obrázku 3.3.



Obr. 3.3: Architektúra aplikácie

Ako je zobrazené na obrázku vyššie, aplikácia je tvorený štyrmi typmi filtrov, z ktorý každý vykonáva rôzne operácie s informáciami prichádzajúcich GPON rámcov. Po spracovaní, každý filter odosiela informácie do komponentu Apache Kafka, kde sú následne informácie uložené pre ďalšie spracovanie v samostatných témach.

3.5.1 Spustenie aplikácie

Aplikácia je na danom zariadení spúšťaná vrámci docker kontajnera. Spustenie aplikácie je realizované spustením Python skriptu *main.py*, napríklad pomocou príkazu 3.13.

Výpis 3.13: Spustenie aplikácie

```
>. python main.py
```

Vo výpise 3.14 je popísaná funkcia *main()* vrámci ktorej dochádza k vytvoreniu inštancie triedy *ProcessFrames* a volaniu metódy *start_threads()*, ktorá zabezpečuje inicializáciu konzumentov, ktorý konzumujú GPON rámce z komponenty Apache Kafka a následné posúvanie prijatých správ na spracovanie do vytvorených filtrov. Táto metóda zároveň rozdelí spracovávanie do separovaných paralelných procesov. Na každej priatej správe sú vykonávané rovnaké úkony spracovania.

Výpis 3.14: Skript main.py

```
"""
Main class - running class
Starts processing of frames
"""

from process_frames.process_frames import ProcessFrames


def main():
    print('Starting processing GPON Frames')
    process_frames: ProcessFrames = ProcessFrames()
    process_frames.start_threads()


if __name__ == "__main__":
    main()
```

3.5.2 Spracovanie rámcov

Trieda *ProcessFrames* obsahuje základne metódy zabezpečujúce spracovávanie rámcov. V konštruktore tejto triedy vytvoríme inštancie tried jednotlivých filtrov a inštanciu producenta, ktorý je následne zdieľaný medzi všetkými inštanciami filtrov. Zdieľanie producenta medzi viacerými procesmi resp. vláknami je odporúčaným spôsobom ako zdieľať prostriedky jedného producenta, v podstate je tento prístup efektívnejší a rýchlejší ako vytvárať pre každý proces unikátnu inštanciu producenta [20].

V rámci tejto triedy sú vytvorené dve metódy tvoriace procesy v dvoch samostatných vláknach. Každý z týchto procesov obsahuje svoju vlastnú inštanciu konzumenta, ktorý je pripojený do rovnakej skupiny konzumentov. Ako bolo spomenuté v texte vyššie, aby bolo možné využiť potenciál paralelného spracovávania dát prostredníctvom Apache Kafka z jednej témy, sú jednotlivé rámce produkované do dvoch partícií. Preto sú v tomto prípade vytvorené dve samostatné procesy s dvomi konzumentmi z ktorých každý odoberá inú partíciu, čím sa zabezpečí paralelná konzumácia dát. Na výpise 3.15 nižšie je zobrazená funkcia pre vytvorenie inštancie konzumenta, návratovou hodnotou tejto funkcie je samotná inštancia.

Takto vytvorený konzument je iterovateľný, preto rámce ktoré obsahuje môžeme získať jeho postupnou iteráciou.

Výpis 3.15: Vytvorenie inštancie konzumenta

```
1  from kafka import KafkaConsumer
2  from json import loads
3
4  def initialize_consumer():
5      consumer = KafkaConsumer('GPONFrames',
6                               bootstrap_servers=[localhost:9092],
7                               auto_offset_reset='earliest',
8                               enable_auto_commit=True,
9                               group_id='GPONFrames-consumers',
10                              value_deserializer=lambda data:
                                    loads(data.decode('utf-8')),
11                              api_version=(0, 10, 1))
12     return consumer
```

Kafka konzument má nastavené nasledujúce parametre:

- GPONFrames – Ako prvý argument je uvedený názov témy, z ktorej chceme správy prijímať resp. konzumovať.
- bootstrap_servers – IP adresa a číslo portu Kafka servera, v tomto prípade sa jedná o adresu docker kontajnera a port na ktorom naslúcha Apache Kafka.
- auto_offset_reset – Tento parameter určuje ako sa ma konzument chovať po výpadku spojenia alebo po jeho vypnutí.
- group_id – ID skupiny konzumentov, v ktorej sa daný konzument nachádza.
- value_deserializer – Metóda na deserializáciu prichádzajúcich dát.
- api_version – Verzia Kafka API.

Po vytvorení inštancie konzumenta, začína konzument prijímať správy zo špecifikovanej témy. Konzument je iterovateľný objekt a vytvára prakticky nekonečný iterovateľný cyklus. Po tom čo je prijatá správa, je táto správa priradená ako argument metódam jednotlivých filtrov, ktoré s touto správou resp. rámcom ďalej manipulujú. Príklad metódy, ktorá bude spustená ako samostatný proces v ktorom sa nachádza inštancia konzumenta a prijaté správy sa posúvajú na ďalšie spracovanie, je zobrazený na výpise 3.16.

Výpis 3.16: Spracovávanie rámcov

```
def process_thread_1(self):
  self.consumer = initialize_consumer()
  self.consumer.assign([TopicPartition('GPONFrames', 0)])
  for message in self.consumer:
    # Filter messages in consumer
    self.connected_onus_filter.filter_connected_onus(
        message.value, self.producer)
    self.ploam_messages_type_filter.filter_ploam_messages_by_type(
        message.value, self.producer)
    self.filter_ploam_messages_by_onu_id.filter_ploam_messages_by_onu_id(
        message.value, self.producer)
    self.unique_ploam_messages_filter.filter_unique_ploam_messages(
        message.value, self.producer)
```

Z výpisu je zreteľné, že vrámci cyklu *for*, ktorým je iterovaný objekt konzumenta, sú volané metódy vytvorených inštancií filtrov spracovávajúce GPON rámce. Argumentami týchto metód sú samotné rámce v podobe JSON objektov a inštancia producenta. V nasledujúcom texte sú hlbšie popísané samotné filtre a spôsob akým rámce spracovávajú a akú podobu a štruktúru majú získané dáta.

Filter aktívnych ONU staníc

Tento filter je tvorený triedou s názvom *FilterConnectedOnus*, metóda v ktorej prebieha spracovávanie dát sa nazýva *filter_connected_onus()* a jej vstupnými argumentami GPON rámec vo formáte JSON a inštancia producenta. Časť zdrojového kódu tejto funkcie je zobrazená na výpise 3.18. Tento filter udržiava a vytvára záznamy o aktívnych a neaktívnych ONU na základe informácií získaných z prijatých rámcov. Štruktúra priateho rámca je popísaná v sekcií 3.2. Po priatí rámca sú z neho získané dve základné informácie a to o aký typ PLOAM správy sa jedná a ID ONU stanice ktorej bola daná správa určená. Ak sa daná ONU stanica nenachádza v zozname unikátnych aktívnych ONU staníc tak je do záznamu aktívnych staníc pridaná

informácia s ID tejto konkrétnej ONU a záznamy sú odoslané do komponenty Apache Kafka. V prípade, že sa stanica nachádza na zozname deaktivovaných ONU, je z tohoto zoznamu vymazaná. Zmena nastáva v situácií ak je prijatá PLOAM správa typu Deactivate_ONU-ID, bližšie popísaná v sekcií 2.3.1. V tomto prípade sú záznamy deaktivovaných ONU staníc aktualizované o stanicu s daným ID, zároveň je stanica vymazaná zo zoznamu aktívnych staníc. Stanica ONU je znovu v záznamoch vedená ako aktívna po tom ako prijme jednu z aktivačných PLOAM správ. Aktivačný proces je bližšie popísaný v sekcií 2.2.4. Po aktualizácií záznamov je tento aktualizovaný záznam odoslaný do Apache Kafka.

Výpis 3.17: Filter aktívnych ONU staníc

```
def filter_connected_onus(self, message, producer):
    message_onu_id: int = message['PLOAMdownstream']['ONUid']
    message_ploam_message_id: int =
        message['PLOAMdownstream']['MessageID']

    if message_onu_id not in self.unique_connected_onus_ids:
        if message_onu_id in self.unique_deactivated_onus_ids:
            self.unique_deactivated_onus_ids.remove(message_onu_id)

        self.unique_connected_onus_ids.append(message_onu_id)
        message: ConnectedOnusMessageFormat =
            ConnectedOnusMessageFormat(message_onu_id)
        data: dict = message.__dict__
        self.buffer['active'].append(data)
        producer.send('ConnectedOnus', value=self.buffer)
        return

    elif message_onu_id in self.unique_connected_onus_ids and
        message_ploam_message_id == PLOAM_DEACTIVATE_MESSAGE:
        self.unique_connected_onus_ids.remove(message_onu_id)
        self.buffer['active'] = [message if message['onu_id'] !=
            message_onu_id else "" for message in self.buffer['active']]
        message: ConnectedOnusMessageFormat =
            ConnectedOnusMessageFormat(message_onu_id)
        data: dict = message.format_message()
        self.buffer['deactivated'].append(data)
        self.unique_deactivated_onus_ids.append(message_onu_id)

        producer.send('ConnectedOnus', value=self.buffer)
        return
```

Odosielané záznamy s informáciami o aktívnych a neaktívnych ONU staniciach majú formát ako na výpise 3.18. Pri informácií o každej jednotke je pridaná taktiež informácia s časovým údajom o aktivácií resp. deaktivácií.

Výpis 3.18: Filter aktívnych ONU staníc

```
{
    'active':[
       {
            'onu_id': 1,
            'date':  2021 -05-11 09:38:04.352605
        },
        {
            'onu_id':   2  ,
            'date':  2021 -05-11 09:38:04.358705
        },
    ],
    'deactivated':[
       {
            'onu_id':   3  ,
            'date':  2021 -05-11 09:3 :04.764705
        }
    ]
}
```

Filter unikátnych PLOAM správ

Tento filter je definovaný triedou s názvom *FilterUniquePloamMessages*, táto trieda okrem iných obsahuje metódu, ktorá zabezpečuje spracovávanie samotných rámcov. Názov tejto metódy je *filter_unique_ploam_messages()*, kde ako v predchádzajúcom prípade sú argumentami GPON rámec a inštancia producenta, ktorý bude spracované dáta odosielať do komponenty Apache Kafka. Zdrojový kód tejto metódy je zobrazený na výpise 3.19.

Výpis 3.19: Filter aktívnych ONU staníc

```
def filter_unique_ploam_messages(self, message, producer: KafkaProducer):
    ploam_message_onu_id = message['PLOAMdownstream']['ONUid']
    ploam_message_id = message['PLOAMdownstream']['MessageID']

    if ploam_message_onu_id not in self.used_ploam_messages_dict.keys():
        self.used_ploam_messages_dict[ploam_message_onu_id] = {}
        self.update_used_messages_count(ploam_message_onu_id,
            ploam_message_id)
        self.buffer[ploam_message_onu_id]: dict = {}
        self.update_buffer(ploam_message_onu_id, ploam_message_id)
        self.update_used_messages_count(ploam_message_onu_id,
            ploam_message_id)
        producer.send('UniquePloamMessages', value=self.buffer)

    else:
        self.update_used_messages_count(ploam_message_onu_id,
            ploam_message_id)
        self.update_buffer(ploam_message_onu_id, ploam_message_id)
        producer.send('UniquePloamMessages', value=self.buffer)
```

V úvode metódy získame zo spracovávaného rámca potrebné údaje a to ONU ID a typ PLOAM správy. Ak daná ONU stanica nie je vedená v záznamoch, je jej vytvorený nový záznam. Následne pomocou metódy *update_used_messages_count()*, zobrazenej na výpise 3.20 aktualizujeme záznamy o počte prijatých PLOAM správ daného typu, patričnou ONU stanicou. Ďalšie metódy používané v rámci tejto témy sú uvedené v prílohe B.2.

Výpis 3.20: Metóda *update_used_messages_count()*

```

def update_used_messages_count(self, ploam_message_onu_id,
    ploam_message_id):
    if self.used_ploam_messages_dict[ploam_message_onu_id].get(
        ploam_message_id, False):
        self.used_ploam_messages_dict[ploam_message_onu_id]
            [ploam_message_id] += 1
    else:
        self.used_ploam_messages_dict[ploam_message_onu_id]
        [ploam_message_id] = 1
```

Po aktualizácií záznamov vedených vrámci aplikácie, sú záznamy odoslané do kom-

ponenty Apache Kafka, konkrétne do témy s názvom *UniquePloamMessages*. Štruktúra dát ukladaných v komponente Apache Kafke je zobrazená na výpise 3.21. Dáta sú vo formáte JSON, každá z aktívne komunikujúcich staníc, v tomto príklade sú to stanice s ONU-ID 255 a 0, obsahuje informácie o množstve, type a ďalších informáciach jednotlivých zachytených správ. Na výpise nižšie môžeme teda vidieť, že stanica s ONU-ID 0 priala celkovo deväť správ typu Ranging_time_message, ktorých ID má decimálnu hodnotu 4.

Výpis 3.21: Štruktúra dát v téme *UniquePloamMessages*

```
1
2  {
3      "255":{
4          "3":{
5              "ploam_message_id": 3,
6              "count": "4",
7              "ploam_message_name": "ASSIGN_ONUID_MESSAGE",
8              "ploam_message_id_hex": 0x03",
9          },
10         "11":{
11          "ploam_message_id": 11,
12             "count": "45",
13             "ploam_message_name": "NO_MESSAGE",
14             "ploam_message_id_hex": "0x0B",
15         },
16     },
17     "0": {
18         "4":{
19             "ploam_message_id": 4,
20             "count": "9",
21             "ploam_message_name": "RANGING_TIME_MESSAGE",
22             "ploam_message_id_hex": "0x04",
23         },
24     }
25 }
```

Filter PLOAM správ podľa typu

V rámci tohoto filtru sú jednotlivé PLOAM správy filtrované a rozdeľované do rôznych tém v komponente Apache Kafka na základe typu PLOAM správy. Každý typ PLOAM správy bude uložený do samostatnej témy, ktorej názov bude zodpove-

dať formátu *PloamType<Ploam_message_type>*. Napríklad PLOAM správy typu Rangin_time, ktorých decimálna hodnota ONU-ID je 4, budú k dispozícií v téme s názvom *PloamType4*. Tento filter je implementovaný ako trieda s názvom *FilterPloamMessagesByType*. Spracovávanie rámcov je vykonávané pomocou metódy *filter_ploam_messages_by_type()* z výpisu 3.22.

Výpis 3.22: Metóda *filter_ploam_messages_by_type()*

```
1
2  def filter_ploam_messages_by_type(self, message, producer):
3          ploam_message_onu_id = message['PLOAMdownstream']['ONUid']
4          ploam_message_id = message['PLOAMdownstream']['MessageID']
5          ploam_message_data = message['PLOAMdownstream']['Data']
6
7          if ploam_message_id not in self.buffer.keys():
8              self.buffer[ploam_message_id] = []
9
10         if ploam_message_id != 11:
11             message = FilterMessagesByTypeFormat(ploam_message_id,
                   ploam_message_onu_id, ploam_message_data)
12             self.buffer[ploam_message_id].append(message.__dict__)
13             producer.send(f'PloamType{ploam_message_id}',
                   value=json.dumps(self.buffer[ploam_message_id]))
```

Po aktualizácií záznamov, sú jednotlivé správy odoslané do topicov podľa názvov opísaných v texte vyššie, kde budú k dispozícií na ďalšie spracovanie. Formát dát uložených v témach je zobrazený na výpise 3.23. Dáta majú štruktúru poľa, ktoré obsahuje množinu JSON objektov, ktoré obsahujú informácie ako ONU-ID, PLOAM-ID, typ PLOAM správy a samotné dáta, ktoré správa obsahuje.

Výpis 3.23: Metóda *Formát dát v témach typu PloamType*

```
1 [
2   {
3      'onu_id':'',
4         'ploam_message_id':'',
5         'ploam_message_name':'',
6         'ploam_message_name_bin':'',
7      'data':''
8   },
9 ]
```

Filter PLOAM správ podľa ONU stanice

Tento filter je funkčne podobný filtru spomenutému v sekcií 3.5.2. Prijaté správy delí do tém podľa ONU-ID cieľovej stanice. Témy majú názov vo formáte *PloamOnuId<ONU-ID>*, teda správy určené pre stanicu ktorej ONU-ID je 4, budú uložené v téme s názvom PloamOnuId4. Trieda tohoto filtra má názov *FilterPloamMessagesByOnuId* a metóda zabezpečujúca spracovávanie správ má názov *filter_ploam_messages_by_onu_id()*. Zdrojový kód tejto metódy je zobrazený vo výpise 3.24.

Výpis 3.24: Metóda *filter_ploam_messages_by_onu_id()*

```
1  def filter_ploam_messages_by_onu_id(self, message, producer:
       KafkaProducer):
2          ploam_message_onu_id = message['PLOAMdownstream']['ONUid']
3          ploam_message_id = message['PLOAMdownstream']['MessageID']
4          ploam_message_data = message['PLOAMdownstream']['Data']
5
6          if ploam_message_onu_id not in self.buffer.keys():
7              self.buffer[ploam_message_onu_id] = []
8
9          if ploam_message_id != 11:
10             message = FilterMessagesByOnuIdFormat(ploam_message_id,
                   ploam_message_onu_id, ploam_message_data)
11             self.buffer[ploam_message_onu_id].append(message.__dict__)
12             producer.send(f'PloamOnuId{ploam_message_onu_id}',
                   value=json.dumps(self.buffer[ploam_message_onu_id]))
```

Formát dát uložených v témach je rovnaký ako na príklade 3.23.

3.6 Získanie spracovaných dát

Spracované dáta filtrami popísanými v sekcií vyššie sú uložené v témach v komponente Apache Kafka. Aplikácie, ktoré budú chcieť tieto dáta využívať alebo ďalej spracovávať, budú musieť implementovať Kafka konzumenta. Príklad takejto implementácie je zobrazený na výpise 3.15, ale môže byť samozrejme vyhotovený v rôznych programovacích jazykoch. Nižšie v tabuľke 3.1 je uvedený zoznam všetkých vytvorených tém s priradenými filtrami, ktoré zabezpečujú spracovávanie dát určené pre danú tému.

Názov témy	Názov filtra
PloamType<PLOAM_Message_Id>	FilterPloamMessagesByType
PloamOnuId<ONU-ID>	FilterPloamMessagesByOnuId
UniquePloamMessages	UniquePloamMessages
ConnectedOnus	FilterConnectedOnus

Tab. 3.1: Prehľad tém a filtrov

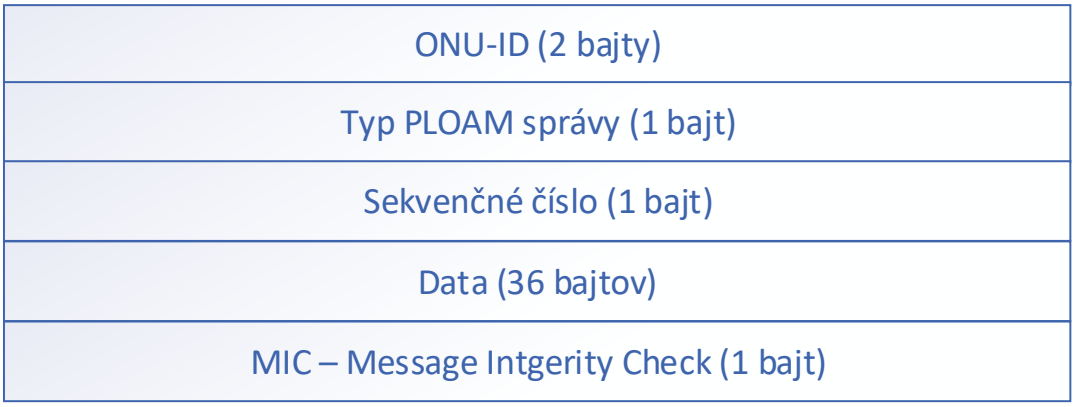
Na obrázku 3.4 nižšie, je zobrazený konzolový výstup skriptu simulujúceho aplikáciu konzumujúcu spracované dáta z témy *PloamType1*. Tieto dáta vo formáte JSON sú taktiež vyobrazené v prílohe B.3. Kde sú združené všetky zachytené správy typu

Obr. 3.4: Konzolový výstup z témy PloamType1

3.7 Využitie systému v aktuálnych štandardoch GPON

Medzi novšie a aktuálnejšie štandardy GPON sú považované napríklad XG-PON a NG-PON2. Štandard XG-PON je často nazývaný aj 10G-PON. Ako názov na-

pomína, tento štandard bol navrhnutý tak, aby jeho maximálne možné prenosové rýchlosti boli 10 Gbit/s. Najnovším štandardom je NG-PON2 (Next Generation PON version 2). Tento štandard je navrhnutý tak, aby dosahoval rýchlosti až 40 Gbit/s [22, 23]. Všetky nové štandardy sú vytvárané aj s cieľom čo možno najlepšej a najjednoduchšej spätnej kompatibility, preto princípy n akých fungujú tieto štandardy nie sú diametrálne odlišné práve naopak, sú si podobné. Všetky tieto štandardy obsahujú servisný a riadiaci komunikačný PLOAM kanál, pomocou ktorého medzi sebou komunikujú OLT a ONU stanice. V rámci tohoto komunikačného si vymieňajú PLOAM správy rôznych typov a význam. Systém popísaný v tejto práci spracováva práve tieto správy zachytené pri komunikácií. Štandardy NG-PON2 a XG-PON majú štruktúru týchto správ podobnú ak nie identickú, správy majú dĺžku 48 bajtov [22]. Táto štruktúra je zobrazená na obrázku 3.5.



Obr. 3.5: Štruktúra PLOAM správ štandardu XG-PON

Avšak GPON PLOAM správy spracovávané vrámci tejto práce majú dĺžku 13 bajtov a odlišnú štruktúru, ktorá bližšie popísaná v sekcií 2.3.1. Túto skutočnosť práve vidím ako prvý a primárny problém, ktorý bude potrebné vyriešiť v prípade využívania systému na monitorovanie sietí vyšších štandardov. V prípade, že dôjde úspešne k úpravám na strane zariadenia na zachytávanie rámcov priamo zo siete a bude možné úspešne zachytiť aj rámce typov XG-PON resp. NG-PON2 bude na strane aplikácie nutné vykonať len minimálne úpravy. Všetky správy sú nejakým spôsobnom spracovávané primárne na základe dvoch parametrov a to ONU-ID a PLOAM_Message_Id a tieto parametre ako môžeme vidieť na vyobrazených štruktúrach obsahujú všetky PLOAM správy naprieč všetkými štandardmi. Z toho vyplýva, že samotná aplikácia, po tom ako bude možné úspešne získavať rámce zo zostupného smeru aj sietí s vyššími štandardmi, bude použiteľná s minimálnymi úpravami aj na spracovávanie rámcov vyšších štandardov. Avšak pri nasadzovaní

aplikácie do systémov na monitorovanie sietí vyšších štandardov treba určite zobrať do úvahy vyššie prenosové rýchlosti v týchto sieťach, čo znamená niekoľko násobný nárast zachytených rámcov za jednotku času. Výkonnostný problém by bolo možné riešiť napríklad ešte rozsiahlejšou paralelizáciou prenosu pomocou viacerých producentov na strane systému na zachytávanie rámcov a viacerých paralelných konzumentov na strane aplikácie.

Záver

Cieľom diplomovej práce bolo do systému na monitorovanie udalostí v pasívnych optických sieťach vhodne implementovať technológiu Apache Kafka ako potenciálne úložisko rámcov a následne vytvoriť aplikáciu implementovanú v jazyku Python, ktorá bude uložené rámce spracovávať a získavať z nich žiadúce informácie. V teoretickej časti práce sú priblížené technológie na distribúciu správ a ich princíp fungovania, krátke porovnanie jednotlivých architektúr na základe priepustnosti a následne opis samotného Apache Kafka. Taktiež sú v tejto časti popísané základné informácie o pasívnych optických sieťach konkrétne o štandarde GPON ako aj informácie o rôznych typoch servisných PLOAM správ.

Systém je tvorený dvom hlavnými komponentami postavenými na architektúre docker kontajnerov. Tieto komponenty fungujú ako nezávislé mikroslužby, kde prvou službou je Apache Kafka a druhou službou je aplikácia na spracovávanie a filtrovanie GPON rámcov. Rámce sú najprv spracované zo vstupných súborov a následne formou JSON objektov odosielané do témy v Apache Kafka. Služba aplikácie načúva a prichádzajúce rámce následne konzumuje a ďalej spracováva. Takto spracované informácie ukladá do vytvorených tém znova v komponente Apache Kafka, kde sú uložené a pripravené na ďalšie spracovanie.

Technológia Apache Kafka sa ukázala ako vhodná komponenta systému na spracovávanie rámcov v pasívnych optických sieťach. Vďaka jej vysokej priepustnosti a flexibilite umožňuje implementáciu viacerých, paralelne fungujúcich konzumentov, ktorý sú schopný prenášané rámce spracovávať nezávisle na sebe. Vďaka robustnosti tejto technológie môže byť monitorovací systém, v prípade potreby, rozšírený o podporu spracovávania GPON rámcov vo vzostupnom smere a taktiež o spracovávanie rámcov sietí X – GPON a to pridaním ďalších filtrov.

Vrámci diplomovej práce bola vytvorená aplikácia, ktorá dokáže zachytávané GPON rámce uchovávať a následne z nich získavať relevantné informácie a tieto informácie znova uchovávať a poskytnúť ich rôznym napríklad zobrazovacím aplikáciam tretích strán.

;

Literatúra

[1] GNATYK, Romana. *Microservices vs Monolith: which architecture is the best choice for your business?* N-iX [online]. 3.11.2018 [cit. 2020-10-10]. Dostupné z: https://www.n-ix.com/microservices-vs-monolith-which-architecture-best-choice-your-business/

[2] THÖNES, Johannes *Microservices in IEEE Software*, vol. 32, no. 1, pp. 116-116, Jan.-Feb. 2015.

[3] GARBARINO, Jimena. *Communicate Between Microservices with Apache Kafka* [online]. 2020 [cit. 2020-10-10].
Dostupné z: https://developer.okta.com/blog/2020/01/22/kafka-microservices

[4] *Powered By* [online]. 2020 [cit. 2020-10-19].
Dostupné z: https://cwiki.apache.org/confluence/display/KAFKA/Powered+By

[5] KUMAR, Manish a Chanchal SINGH.*Building Data Streaming Applications with Apache Kafka* [online]. Birmingham, 35 Livery Street, B3 2PB, UK: Packt Publishing, 2017 [cit. 2020-10-12]. ISBN 978-1-78728-398-5.
Dostupné z: https://books.google.cz/books?id=ZpZGDwAAQBAJprintsec=frontcoverhl=csv=onepageqf=false

[6] NARKHEDE, Neha, Gwen SHAPIRA a Todd PALINO. *Kafka: The Definitive Guide* [online]. 1005 Gravenstein Highway North, Sebastopol, CA 95472, United States of America: O'Reilly Media, 2017 [cit. 2020-10-12]. ISBN 978-1-491-99065-0.
Dostupné z: https://www.confluent.io/resources/kafka-the-definitive-guide/

[7] CHEN, Chen, Yoav TOCK, Hans-Arno JACOBSEN a Roman VITENBERG. *Weighted Overlay Design for Topic-Based Publish/Subscribe Systems on Geo-Distributed Data Centers* [online]. 23.06.2015 [cit. 2020-10-17].
Dostupné z: doi:10.1109/ICDCS.2015.55

[8] *What Is Pub/Sub* [online]. Google Cloud, 2020 [cit. 2020-10-17].
Dostupné z: https://cloud.google.com/pubsub/docs/overview

[9] *Apache Kafka Documentation* [online]. 2017 [cit. 2020-10-19].
Dostupné z: https://kafka.apache.org/documentation/

[10] *What is Apache Kafka?* [online]. 2020 [cit. 2020-10-19].
Dostupné z: https://aws.amazon.com/msk/what-is-kafka/

[11] *Kafka vs. Pulsar vs. RabbitMQ: Performance, Architecture, and Features Compared* [online]. 2020 [cit. 2020-10-19].
Dostupné z: https://www.confluent.io/kafka-vs-pulsar/

[12] *Optické přístupové sítě* [online]. 2020 [cit. 2020-11-15].
Dostupné z: https://publi.cz/books/185/11.html

[13] ITU-T G.984.1 *Gigabit-capable passive optical networks (G-PON): General characteristics.* [online]. Ženeva, Švajčiarsko, 2008 [cit. 2020-11-15].
Dostupné z: https://www.itu.int/rec/T-REC-G.984.1-200803-I/en

[14] HOOD, Dave a Elmar TROJER. *GIGABIT-CAPABLE PASSIVE OPTICAL NETWORKS.* Hoboken, New Jersey: John Wiley Sons, 2012. ISBN 978-0-470-93687-0.

[15] *GPON Fundamentals* [online]. 2020 [cit. 2020-10-19].
Dostupné z: https://sites.google.com/site/amitsciscozone/home/gpon/gpon-fundamentals

[16] ITU-T G.984.3 *Gigabit-capable passive optical networks (G-PON): Transmission convergence layer specification* [online]. Ženeva, Švajčiarsko, 2008 [cit. 2021-04-15].
Dostupné z: https://www.itu.int/rec/T-REC-G.984.3-201401-I/en

[17] BERNÁT, Ivan. Docker a jeho použitie pri kontajnerizácii. *Magazín KPI* [online]. 2020 [cit. 2020-10-19].
Dostupné z: https://magazin.kpi.fei.tuke.sk/2019/02/docker-a-jeho-pouzitie-pri-kontajnerizacii/

[18] HELVICK, Tom. Docker Fundamentals: Introduction to an Increasingly Essential Developer Tool. *Intertech* [online]. 2020 [cit. 2020-10-19].
Dostupné z: https://magazin.kpi.fei.tuke.sk/2019/02/docker-a-jeho-pouzitie-pri-kontajnerizacii/

[19] AZAR, Jean-Paul. Kafka Topic Architecture - Replication, Failover, and Parallel Processing. *DZone* [online]. [cit. 2021-5-7]. Dostupné z: https://dzone.com/articles/kafka-topic-architecture-replication-failover-and

[20] Kafka-Python Docs *KafkaProducer* [online]. Cit. 2021-04-19.
Dostupné z: https://kafka-python.readthedocs.io/en/master/apidoc/KafkaProducer.html

[21] Kafka-Python Docs *KafkaConsumer* [online]. Cit. 2021-04-19.
Dostupné z: https://kafka-python.readthedocs.io/en/master/apidoc/KafkaConsumer.html

[22] HORVATH, Tomas, Petr MUNSTER, Vaclav OUJEZSKY a Josef VOJTECH. Activation Process of ONU in EPON/GPON/XG-PON/NG-PON2 Networks. Applied Sciences [online]. 2018, 8(10) [cit. 2021-5-13]. ISSN 2076-3417. Dostupné z: doi:10.3390/app8101934

[23] ITU-T G.989.3 *40-Gigabit-capable passive optical networks (NG-PON2): Transmission convergence layer specification* [online]. Ženeva, Švajčiarsko, 2016 [cit. 2021-05-10].
Dostupné z: https://www.itu.int/rec/T-REC-G.989.3/en

Zoznam symbolov, veličín a skratiek

API	Aplication Interface
CRC	Cyclic Redundancy Check
FIFO	First In First Out
GPON	Gigabyte Capable PON
GTC	GPON Transmission Layer
GEM	GPON Encapsulation Method
HEC	Hybrid Error Correction
IP	Internet Protocol
ITU-T	ITU Telecommunication Standardization Sector
IoT	Internet of Things
JSON	Javascript Object Notation
LOF	Lost Of Frames
ONU	Optical Network Unit
ONT	Optical Network Terminal
OLT	Optical Line Terminator
PCBd	Physical Control Block downstream
PLOu	hysical Layer Overhead upstream
PLI	Payload Length Indicator
PLOAM	Physical Layer Operations And Maitenance
PON	Passive Optical Networks
PS	Publish-Subscribe
PTP	Point-to-Point
REST	Representational State Transfer
SDU	Service Data Unit

SQL	Structured Query Language
TCP	Transimision Control Protocol
WDD	Wavelength Division Multiplex
XML	Extensible Markup Language
XG–PON	10 Gigabit PON

4 Obsah elektronických príloh

V tejto kapitole je popísaný obsah a štruktúra elektronických príloh. Elektronická príloha obsahuje zdrojové súbory komponentov systému na monitorovanie udalostí v pasívnych optických sieťach, informačný dokument a text vypracovanej diplomovej práce.

- `/` ..... adresár priložených elektronických úloh
  - `README.txt` ..... logotypes
  - `xpanca00_text_prace.pdf` ..... other graphic files
  - `dp_gpon_monitoring_system_kafka` ..... Zdrojové súbory Apache Kafka
  - `dp_gpon_monitoring_system_application` ..... Zdrojové súbory aplikácie

Zoznam príloh

A Formát prenášaných GPON rámcov

Výpis A.1: Formát prenášaných rámcov

```
{
    "1":{
        "Psync":3064672736,
        "Identification":{
            "FEC":1,
            "Reserved":0,
            "SuperframeCounter":231583400
        },
        "PLOAMdownstream":{
            "ONUid":255,
            "MessageID":11,
            "Data":"AAAAAAAAAAAAAA==",
            "CRC":158
        },
        "BIP":253,
        "Plend":[
            {
                "Blen":0,
                "Alen":0,
                "CRC":0
            },
            {
                "Blen":0,
                "Alen":0,
                "CRC":0
            }
        ],
        "Bwmap":[]
    }}
```

B Zdrojové kódy aplikácie

B.1 Skript producent.py

Výpis B.1: Producent

```
from json import dumps
from kafka import KafkaProducer
import glob
import json
import ast

# Path to stored Dataset
path = r"D:\gpon_frames\*.txt"
files = glob.glob(path)

# Create producer instance with following configuration
producer = KafkaProducer(bootstrap_servers=['localhost:9092'],
                         value_serializer=lambda v:
                             json.dumps(v).encode('utf-8'),
                         api_version=(0, 10, 1))

# Iterate through all files from folder
for file in files:
    with open(file, 'r') as f:
        response = json.dumps(f.read())
        response = ast.literal_eval(json.loads(response))

        # Iterate through JSON objects (frames)
        for i in range(1, len(response)):
            data = {i: response[i]}
            producer.send('GPONFrames', value=data)
```

B.2 Metóda *update_buffer()*

Výpis B.2: Metóda *update_buffer()*

```
1
2  def update_buffer(self, ploam_message_onu_id, ploam_message_id):
3          if self.buffer[ploam_message_onu_id].get(ploam_message_id, False)
               is False:
4              counter =
                   self.get_ploam_message_type_count(ploam_message_onu_id,
                   ploam_message_id)
5
6              # Create new json data object with information about current
                   ploam message
7              message = UniquePloamMessagesFormat(
8                  ploam_message_id,
9                  ploam_message_onu_id,
10                 counter
11             )
12             self.buffer[ploam_message_onu_id][ploam_message_id]: dict = {}
13             self.buffer[ploam_message_onu_id][ploam_message_id] =
                   message.__dict__
14
15         else:
16             # Get message count
17             count =
                   self.get_ploam_message_type_count(ploam_message_onu_id,
                   ploam_message_id)
18             self.buffer[ploam_message_onu_id][ploam_message_id]['counter']
                   = count
```

B.3 Výstup témy PloamType1

Výpis B.3: Výstup témy PloamType1


```
[
   {
      "onu_id":255,
      "ploam_message_id":1,
      "data":"IAAAqqtZgyAAAA==",
      "ploam_message_name":"UNKNOWN TYPE",
      "ploam_message_id_bin":"UNKNOWN TYPE"
   },
   {
      "onu_id":255,
      "ploam_message_id":1,
      "data":"IAAAqqtZgyAAAA==",
      "ploam_message_name":"UNKNOWN TYPE",
      "ploam_message_id_bin":"UNKNOWN TYPE"
   },
   {
      "onu_id":255,
      "ploam_message_id":1,
      "data":"IAAAqqtZgyAAAA==",
      "ploam_message_name":"UNKNOWN TYPE",
      "ploam_message_id_bin":"UNKNOWN TYPE"
   },
   {
      "onu_id":255,
      "ploam_message_id":1,
      "data":"IAAAqqtZgyAAAA==",
      "ploam_message_name":"UNKNOWN TYPE",
      "ploam_message_id_bin":"UNKNOWN TYPE"
   },
   {
      "onu_id":255,
      "ploam_message_id":1,
      "data":"IAAAqqtZgyAAAA==",
      "ploam_message_name":"UNKNOWN TYPE",
      "ploam_message_id_bin":"UNKNOWN TYPE"
   }
]
```